UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Simulador del Perceptrón Multicapa

Autor: Antonio de la Mata Martín

Tutora: Inés Mª Galván León

Agradecimientos

A todas aquellas personas que me han acompañado a lo largo de este viaje,

en especial a Elena y a Inés por su apoyo, paciencia y dedicación.

Resumen

El Perceptrón Multicapa es una Red de Neuronas Artificiales ampliamente utilizada en la actualidad para la resolución de problemas de diversas clases, entre los cuales se pueden encontrar problemas de clasificación, regresión y de predicción de series temporales.

La independencia del contexto de las Redes Neuronas hace que su uso aporte soluciones en numerosos campos de estudio, lo cual convierte a las Redes de Neuronas en una herramienta interesante a lo hora de afrontar la resolución de problemas.

A lo largo del presente trabajo se documenta y detalla el desarrollo de un simulador implementado en Java que emula el funcionamiento del Perceptron Multicapa. En las diferentes secciones que se desarrollarán a continuación se expondrán de manera detallada las diferentes fases y actividades asociadas al desarrollo del simulador, partiendo del estudio de los fundamentos teóricos en que se basa el Perceptrón Multicapa, hasta llegar a la realización de las distintas pruebas para su validación.

El principal objetivo que motiva el desarrollo de dicho simulador, así como del presente trabajo, es ofrecer una herramienta sencilla y eficaz que facilite el acceso y el uso del Percpetron Multicapa al público en general. Además su desarrollo contempla su utilización para uso docente en ámbitos universitarios.

En el trabajo se realiza también un estudio preliminar de la capacidad del Perceptron Multicapa (aprovechando el simulador desarrollado) para predecir la producción de huevos en granjas avícolas a partir de la edad de las aves y producciones en días anteriores. Este punto de vista es novedoso en el contexto de granjas avícolas y tiene diferentes aplicaciones prácticas enfocadas a la optimización de costes.

Palabras clave:

Redes de Neuronas Artificiales, Perceptrón Multicapa, Algoritmo de retro-propagación, Java, Software libre.

Abstract

The Multilayer Perceptron is an Artificial Neural Network widely used to solve problems of various kinds, including classification, regression and time series prediction problems among others.

Context independence of Artificial Neuronal Networks makes its use provide solutions in many fields of study; this makes Neural Networks an interesting tool when dealing with problem solving.

This paper documents and details the development of a simulator implemented in Java which emulates the behavior of the Multilayer Perceptron. The different phases and activities associated with the development of the simulator will be presented in detail in the various sections to be developed, from the study of the theoretical foundations underlying the Multilayer Perceptron, up to the completion of the various validation tests.

The main objective that motivates the development of this simulator, as well as the present paper is to provide a simple and effective tool that facilitates the access and use of the Multilayer Perceptron to the general public. It is also considered its use for academic purposes in universities.

This paper also performs a preliminary study of the ability of the Multilayer Perceptron (using the simulator developed) to predict the egg production in a poultry farm from data such as the birds' age and the productions in previous days. This view is novel in the context of poultry farms and has practical applications focused on cost optimization.

Keywords:

Artificial Neural Networks, Multilayer Perceptron, Back-Propagation Algorithm, Java, Free Software.

Índice:

Índice de Ecuaciones:

Índice de tablas:

Índice de ilustraciones:

Capítulo 1. Introduction

1.1. Introduction to Artificial Neural Networks

Artificial neural networks emerge in the twentieth century along the forties, when neurologists as McCulloch and Pitts proposed the first models of neural networks; Years later Donald Hebb worked in various ideas about neuronal learning that lead to Hebb rule.

In the late fifties the first models of neural networks with practical applications in industry appeared, like the Simple Perceptron and the ADALINE. In 1986 Rumelhart and other researchers presented the Generalized Delta Rule in order to adapt weights back propagating the errors (BackPropagation) to multiple layers and the use of nonlinear activation functions. It is from this point that there was a great development in the field of artificial neural networks, appearing some of the major neural networks, such as the Multilayer Perceptron or radial basis networks.

Artificial Neural Networks (RNAs/ANNs hereafter) are mathematical models based on natural biological systems which are tailored to be simulated on conventional computers. The ANNs are therefore computation models inspired on biological neural networks which simulate the nervous systems structure, which are characterized by being composed of interconnected neurons. As happens in Biological Neural Networks, artificial neuron is the basic unit of processing of ANNs.

The structure of the artificial neuron (illustration 1) is similar to the structure of the biological neuron, having several inputs information and computing a single output, which correspond respectively to the dendrites and the axon of the biological neuron.

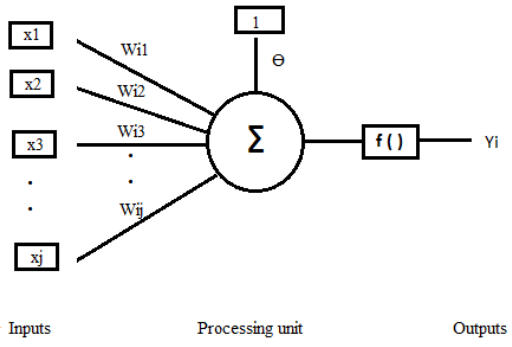

Ilustración 1: Structure of an artificial neuron.

The above illustration shows the structure of an artificial neuron, which has multiple inputs that can come from the outside world or from other neurons and a single output whose value is computed based on the same inputs.

The ANNs are formed from sets of artificial neurons connected by arcs, which are called connections, whose aim is to connect the outputs of the neurons to the inputs of others. Generally these neurons are arranged in layers at different levels.

There are numerous ANNs architectures which respond to various classifications, some of the most common ANNs classified according to its topology (connection patterns between neurons) are:

- Feedforward networks or acyclic:
    - Monolayer: such as the Simple Perceptron or the ADALINE.
    - Multilayer: such as the Multilayer Perceptron and the Radial Basis Network.
- Recurrent networks: networks that have at least one cycle, such as the Hopfield networks or the Boltzmann machine.

The ANNs have different ways of carrying out the learning process, they can do it supervised or unsupervised. In supervised learning it is required that the input data set is previously classified or that its answers are known, while the unsupervised learning knowledge of this output is not required.

The ANNs are used for solving classification problems, regression and time series prediction, among others. The reason why they are used is that the ANNs are characterized by an adaptive learning, because of this they are able to learn to perform tasks based on specific training and then adapt to change as new information is received. They are characterized also because they are able to learn relationships from examples or patterns that represent the problem.

1.2. Project objectives

With this Final Year Project is intended, as the main goal, to build a simulator of neural networks for the Multilayer Perceptron in order to facilitate a tool for teaching purposes in the field of neural networks. This simulator will incorporate all the aspects involved in the creation and training of Multilayer Perceptron. The functions to be performed by the simulator are:

- Defining and modify architecture.
- Defining the activation function of the output neurons of the network.
- Loading training and test files.
- Defining the parameters for the network training.
- Training a network.
- Storing network architectures.
- Loading network architectures.

- Storing the results of the network.

In addition to this primary goal, during the planning and development of the simulator, other objectives were also taken into account. The simulator is intended to be a simple tool, easy to use, which does not require an extensive prior knowledge of neural networks, which have a comfortable and pleasant interface and which does not require a difficult installation, thereby facilitating the user its use and portability.

1.3. Project motivation

The ANNs are a branch of Artificial Intelligence which Computer science commonly uses as one of the possible tools to provide efficient solutions to problems of various kinds such as classification, approximation or prediction in many different fields of study, including medicine, economics, engineering, etc…

Over the years and after the realization of various projects based on these problems, I always found myself with the same problem on trying to apply the ANNs. Such complication arises from the difficulty on finding free and efficient tools which also are intuitive and simple to use and which do not require a complex installation.

In light of this situation, arise the idea and the motivation required for the development of a new tool that fulfilled those requirements in order to provide a mean which is convenient, efficient and easy to use in order to develop solutions based on ANNs and, in particular, using the Multilayer Perceptron. This mentioned tool would provide access to such solutions, but it would not be made only for people with high knowledge in ANNs but, as far as possible would facilitate for people without previous knowledge about their structure and functioning to be able to have access to them; taking advantage of the benefits that ANNs can provide to the resolution of these kind of problems.

For the launch of a new simulator, as well as for the development of any application, it is required to carry out a previous study of similar existing tools, which will provide a wider knowledge on the functionalities that this application should accomplish. At present, there exist various tools which allow the simulation of neural networks architectures and more specifically, the Multilayer Perceptron.

SNNS and its Java version JavaNNS are free access neural network simulators which allow the generation of architectures and learning in order to obtain a series of results depending on the type of architecture and the established patterns. The SNNS version for Linux works in a correct and intuitive way for the Multilayer Perceptron, however its usage is complex for other architectures and it also has the disadvantage of having a very complex installation process. On the other hand, the main advantage of Java's version is that the installation process is very simple and it does not require an environment with high specifications for its execution, nevertheless its usage is unintuitive and its main disadvantage regarding the Multilayer

Perceptron is that it initializes the network's weights to zero, which makes that often the learning is not achieved successfully.

WEKA is a free Access application which provides the users with the tools required to solve classification and approximation or regression problems by using different methodologies, such as classification trees, Bayesian networks or the Multilayer Perceptron among others. WEKA is a widely used tool which has wide recognition in the Artificial Intelligence field, but regarding the Multilayer Perceptron it has some differences. As well as it happens with SNNS and JavaNNS, its usage is unintuitive, besides, its main disadvantage is that it does not show the evolution of the error along with the learning process; it only displays the final error, which hinders the definition of the learning parameters.

NeuroSolutions is asoftware intended for solving problems with high complexity by using different methodologies, ANNs among them. This tool presents a convenient and intuitive interface, but its large amount of options obstructs its usage for inexperienced users, as offering a wide variety of methodologies, both in the field of neural networks as in other fields of study, the use of a specific methodology requires wide knowledge about the different alternatives offered by this tool. However, the main inconvenience of this tool is its high price, which does not make it accessible for everybody.

Neural Network Toolbox is a Matlab library which allows the simulation of different neural networks architectures, but as it happens with NeuroSolutions it has a high cost due to the fact that it requires the purchase of licenses for the use of Matlab, besides, it also requires Matlab programming skills by the user.

There also exists widely used libraries such as FastArtificial Neural Network (FANN) programmed in C or Neuroph in Java, but use of these libraries implies that the user must perform prior programming work

1.4. Document structure

This document forms the theoretical basis and the structure on which the simulator is based. The document shows the following structure:

The current chapter gives a brief introduction to the ANNs, along answering to two important questions, what is to be achieved by performing this job? And why taking this initiative?

The second chapter details the theoretical foundations of Multilayer Perceptron, upon which is founded the simulator.

The third chapter provides in-depth analysis of the system, which specifies the requirements and constraints that it must satisfy and a study of possible alternatives to the solution which has been chosen.

The fourth chapter studies the system design, including its architecture, subsystems and the different classes that make up the application.

The fifth chapter describes a series of experimental tests on different domains in which the proper functioning of the simulator can be analyzed as well as the results obtained.

The sixth chapter includes the user manual of the simulator, which will be provided in order to facilitate its use. The manual describes how to use the various features and options offered by the simulator.

The seventh chapter details the planning for the application development, which includes an analysis of the different phases of the project and the budget for its development.

The eighth chapter explains the different conclusions that have been obtained during the development of this paper as well as the possible future lines of work in case someone wishes to continue to expand the capabilities of this simulator. It also includes an analysis of the difficulties encountered during its development.

After the eighth and final chapter, a series of definitions and acronyms has been included in order to facilitate compression of the document and the bibliography used for this document.

Finally at the end of the document two appendices have been included with the introduction and conclusions chapters in Spanish.

Capítulo 2. El Perceptrón Multicapa

En el presente capítulo se explicará en detalle el Perceptrón Multicapa, la red de neuronas sobre la que trata este trabajo y para la cual se va a desarrollar el simulador. Se explicarán los fundamentos teóricos sobre los que se basa el Perceptrón Multicapa y que, por lo tanto, conforman la base teórica de este Trabajo de Fin de Grado y del simulador. Esta base teórica resulta de gran importancia para conocer y comprender el funcionamiento y la estructura del Perceptrón Multicapa, lo cual resulta imprescindible para llevar a cabo un correcto desarrollo del simulador y es de gran utilidad para comprender el funcionamiento interno de las diferentes funcionalidades del mismo.

2.1. Arquitectura del Perceptrón Multicapa

El Perceptrón Multicapa es un tipo de RNA que surge en 1986 cuando Rumelhart presenta la Regla Delta Generalizada, la cual permite adaptar los pesos de las neuronas de la red propagando los errores hacia atrás. De esta forma, se puede comenzar a trabajar con redes que cuenten con múltiples capas y que utilicen funciones de activación no lineales. Esto permite acabar con las limitaciones de las RNAs que existían hasta el momento, como son el Perceptrón Simple y el ADALINE, las cuales no podían resolver problemas no lineales.

En 1991 Park y Sandberg demuestran que el Perceptrón Multicapa es un aproximador universal. Es a partir de este momento cuando su uso se generaliza y se convierte en una de las arquitecturas de RNAs más utilizadas.

El Perceptrón Multicapa es una arquitectura de RNA que se caracteriza por estar formada de múltiples capas que pueden estar compuestas por una o más neuronas. En el Perceptrón Multicapa se distinguen tres tipos de capa:

- Capa de entrada: el Perceptrón Multicapa tiene una única capa de entrada que está formada por neuronas que representan las entradas de la red. Estas neuronas no actúan como neuronas propiamente dichas (según la definición que se dio en la introducción de neurona artificial), sino que simplemente se encargan de recibir las señales de entrada y propagarlas a la siguiente capa.

- Capas ocultas: el Perceptrón Multicapa tiene al menos una capa oculta, estas capas realizan un procesamiento no lineal de los patrones recibidos.

- Capa de salida: el Perceptrón Multicapa tiene una única capa de salida que está formada por neuronas que representan las salidas de la red y cuya función es proporcionar al exterior la respuesta de la red para cada patrón de entrada.

Por tanto, el Perceptrón Multicapa está compuesto de un mínimo de tres capas, una capa de entrada, una capa de salida y al menos una capa oculta (ilustración 2). El Perceptrón Multicapa es una red "feedforward", lo cual significa que las capas se conectan entre sí, de modo que las salidas de las neuronas de una capa se propagan a las neuronas de la siguiente capa donde actúan como entradas.

En el Perceptrón Multicapa cada neurona está conectada a la entrada de todas las neuronas de la siguiente capa. Las conexiones entre estas neuronas llevan asociado un número real, el cual se conoce como peso de la conexión. Los pesos se designan con la nomenclatura W_{ij}^{c} donde c representa la capa de la que parte, el subíndice i la posición que ocupa la neurona de origen en la capa c y el subíndice j posición que ocupa la neurona de destino en la capa $c+1$ (ilustración 2).

Todas las neuronas, independientemente de la capa en la que se encuentren, a excepción de la capa de entrada, tienen un número real asociado conocido como Umbral de la neurona. Dicho parámetros se considera como un peso más de la neurona cuya entrada es siempre igual a 1; se designa con la nomenclatura u_i^c donde c es la capa a la que pertenece la neurona, el subíndice i es la posición que ocupa dicha neurona en la capa c (ilustración 2). Las salidas o activaciones de estas neuronas se designan con la nomenclatura a_i^c donde c es la capa e i la posición que la neurona ocupa en c.

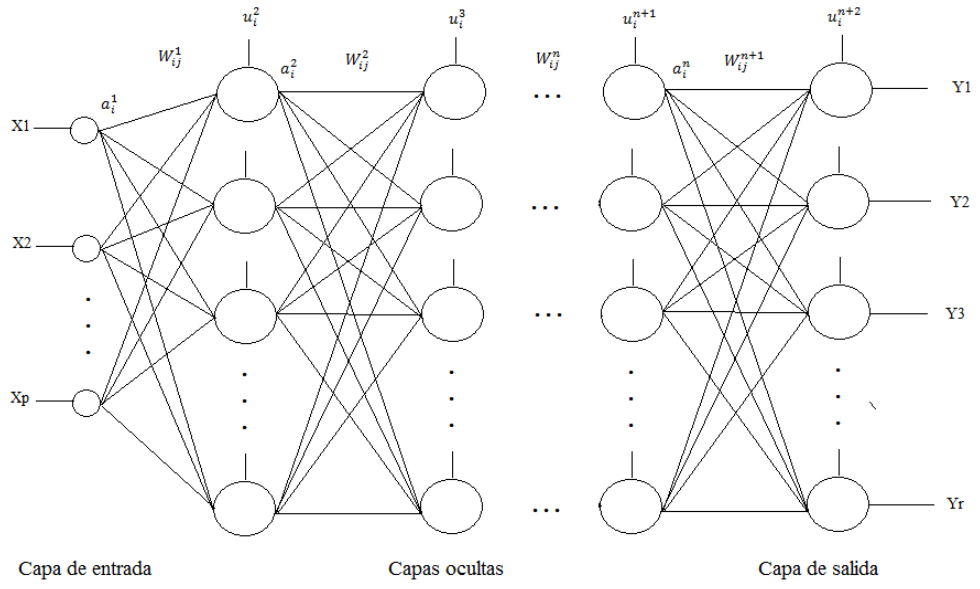

Ilustración 2: Arquitectura del Perceptrón Multicapa

Debido a que el Perceptrón Multicapa es una red "feedforward" se define una relación entre las variables de entrada y las variables de salida de la red. Esta relación se debe a que las variables de entrada se van propagando a lo largo de las diferentes capas de la red hasta llegar a la capa de salida. Para ello, cada neurona del Perceptrón Multicapa procesa de manera

independiente la información que recibe en sus entradas y produce una salida o activación que propaga a la siguiente capa.

Las salidas del Perceptrón Multicapa se corresponden con las activaciones de las neuronas de la capa de salida. Éstas se hallan mediante el cálculo ordenado de las activaciones de sus neuronas propagando las variables de entrada desde la entrada hasta la salida: en primer lugar, se calculan las activaciones de las neuronas de la capa de entrada y se continúa siguiendo el orden de las capas de izquierda a derecha. Las ecuaciones con las que se calculan las activaciones de las neuronas en el Perceptrón Multicapa son las siguientes:

- Activación de las neuronas de la capa de entrada:

$$a_i^1 = x_i \; para \; i = 1, 2, 3, \dots, p$$

Ecuación 1: Calculo de las activaciones en la capa de entrada.

Donde x_i representa el vector o patrón de entrada $X = (x_1, x_2, x_3, \dots, x_p)$.

- Activación de las neuronas de las capas ocultas y de salida:

$$a_i^c = f\left(\sum_{j=1}^{n_{c-1}} (w_{ji}^{c-1} * a_j^{c-1}) + u_i^c\right)$$

Ecuación 2: Calculo de las activaciones de las capas ocultas y de salida.

donde c es la capa donde se encuentra la neurona i cuya activación se está calculando, $c-1$ es la capa anterior, n_{c-1} neuronas que tiene esta capa, a_j^{c-1} es la activación de la neurona j de la capa $c-1$, w_{ji}^{c-1} es el peso que une a la neurona j de la capa $c-1$ con la neurona i de la capa c, u_i^c es el Umbral asociado a la neurona cuya activación se está calculando y f es la llamada función de activación.

Como se puede observar en la ecuación anterior (ecuación 2) para el cálculo de las activaciones de las neuronas ocultas y de salida del Perceptrón Multicapa, además de los cálculos que se realizan sobre las entradas y los pesos asociados a éstas, se les aplica una función f denominada función de activación de la neurona.

Por lo general, la función de activación que se aplica a las neuronas del Perceptron Multicapa es común para todas las neuronas de las capas ocultas y neuronas de salida. En algunas ocasiones, las neuronas de salida aplican función de activación lineal, mientras que las neuronas ocultas aplican funciones no lineales.

Las funciones de activación más comunes y que se pueden utilizar tanto para las neuronas de las capas ocultas, como para las neuronas de la capa de salida, son las siguientes:

- Función de activación sigmoidal: es una función creciente con dos niveles de saturación 0 para el mínimo y 1 para el máximo.

$$f(x) = \frac{1}{1 + e^{-x}}$$

Ecuación 3: Función de activación sigmoidal.

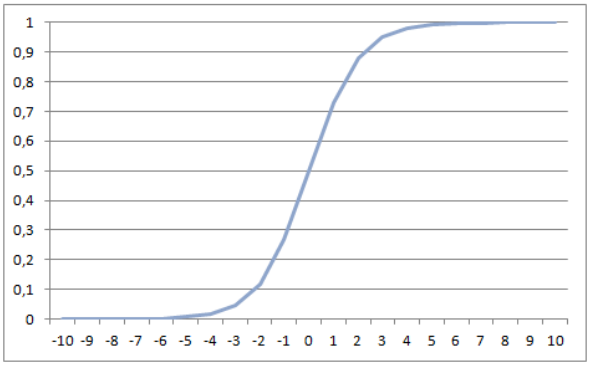

Ilustración 3: Gráfica de la función de activación sigmoidal.

- Función de activación hiperbólica: es una función creciente con dos niveles de saturación -1 para el mínimo y 1 para el máximo.

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Ecuación 4: Función de activación hiperbólica.

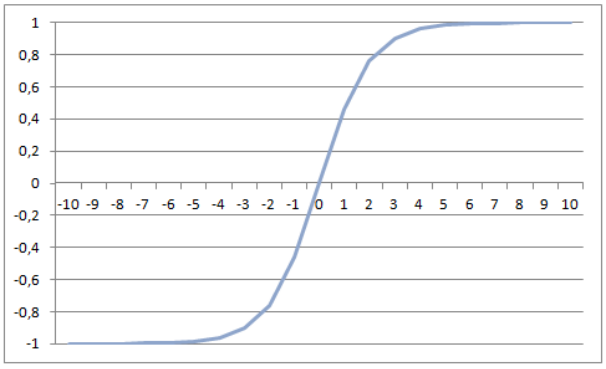

Ilustración 4: Gráfica de la función de activación hiperbólica.

El comportamiento de ambas funciones es equivalente (se observa en la gráfica un mismo comportamiento), generalmente se utiliza la primera (sigmoidal ilustración 3), pero que el uso de una u otra depende del rango en el que estén normalizados los datos.

Como se ha mencionado anteriormente, en ocasiones las neuronas de salida aplican una función de activación lineal, lo cual es una función creciente sin niveles de saturación y que viene dada por:

$$f(x) = x$$

Ecuación 5: Función de activación lineal.

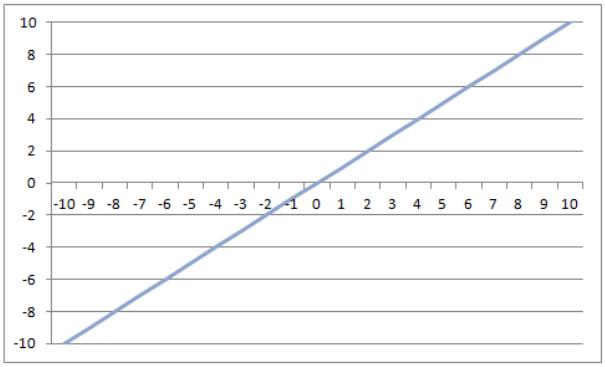

Ilustración 5: Gráfica de la función de activación lineal.

A la hora de definir una arquitectura de Perceptrón Multicapa para su uso en la resolución de un problema, el diseñador deberá tener en cuenta los siguientes factores:

- El número de neuronas de la capa de entrada ha de coincidir con el número de variables de entrada del problema.
- El número de neuronas de la capa de salida ha de coincidir con el número de variables de salida del problema.
- La función de activación se seleccionará en función del recorrido que se desee que tenga la salida, por ejemplo si se desea que la salida tome valores en el rango [0,1] se utilizara la función de activación sigmoidal (ecuación 3).
- No existe un método o regla que determine el número óptimo de capas y neuronas ocultas para la resolución de un problema. Generalmente se establecen realizando un conjunto de experimentos variando dicho número y realizando un análisis posterior de los resultados obtenidos.

  En la actualidad existen líneas de investigación para la determinación automática del número óptimo de capas y neuronas ocultas, algunos de estos trabajos se basan en la aplicación de técnicas evolutivas para estudiar el espacio de posibles arquitecturas.

2.2. Algoritmo de aprendizaje

Se conoce como algoritmo de aprendizaje de una red de neuronas al proceso mediante el cual se adaptan los diferentes parámetros de la red, tales como pesos y umbrales. En el Perceptrón Multicapa este algoritmo se conoce como Algoritmo de RetroPropagación (BackPropagation) o Regla Delta Generalizada.

El Algoritmo de RetroPropagación es un algoritmo de aprendizaje supervisado, su objetivo es minimizar las diferencias entre las salidas obtenidas por la red para una serie de patrones y sus salidas previamente conocidas. Se denomina Algoritmo de RetroPropagación debido a que los errores medidos en la capa de salida se propagan hacia atrás, es decir desde la salida hacia la capa de entrada, adaptando durante este proceso los pesos y umbrales de la red.

Dado que el objetivo de este algoritmo es minimizar las diferencias entre una serie de salidas previamente conocidas y las salidas obtenidas por la red, el problema del aprendizaje en el Perceptrón Multicapa se puede resumir en un problema de minimización del error. Este problema se puede formular de la siguiente manera:

$$Min_W E$$

Ecuación 6: Problema del aprendizaje como problema de minimización del error.

donde W es el conjunto de parámetros de la red, es decir los pesos y los umbrales, y E es la función utilizada para calcular el error.

En el Perceptrón Multicapa por lo general se utiliza como medida del error, el error medio por patrón, el cual se calcula mediante la siguiente función:

$$E = \frac{1}{N}\sum_{n=1}^{N} e(n)$$

Ecuación 7: Función de error medio.

donde N es el número de patrones, y $e(n)$ es el error calculado para cada patrón. Dicho error se calcula mediante la función del error cuadrático:

$$e(n) = \frac{1}{2}(s_i - y_i)^2$$

Ecuación 8: Error cuadrático.

donde s_i es la salida deseada para el patrón i y y_i es la salida obtenida para dicho patrón.

La presencia de funciones de activación no lineales implica que la salida obtenida por la red no sea lineal respecto a los parámetros de ésta, por lo que este problema de minimización del error es un problema no lineal, ante lo cual se deben utilizar técnicas de optimización no lineales en su resolución. La regla Delta Generalizada o Algoritmo de Retropropagación hacen uso del "Método de descenso del gradiente", en particular el método del descenso del gradiente estocástico.

Mediante la aplicación de este método, cada parámetro W de la red es modificado para cada patrón de acuerdo a la siguiente ecuación:

$$w(n) = w(n-1) - \alpha \frac{\partial e(n)}{\partial w}$$

Ecuación 9: Ley de aprendizaje, método del descenso del gradiente.

donde $e(n)$ es el error obtenido para el patrón n (ecuación 8) y α la tasa de aprendizaje.

El cambio en los pesos es proporcional a la derivada del error, la tasa de aprendizaje α es una constante de proporcionalidad que pondera dicha derivada. La tasa de aprendizaje es, por tanto, el parámetro que controla cuánto se desplazan los pesos de la red en la superficie del error y puede tomar valores entre 0 y 1. Para valores altos de la razón de aprendizaje la convergencia del algoritmo es más rápida, teniendo el riesgo que el método puede oscilar alrededor del mínimo. Si por el contrario α es pequeña, la convergencia es más lenta y puede caer en un mínimo local. Generalmente, es necesario realizar una serie de experimentos para conocer la razón de aprendizaje más adecuada para cada problema.

Aplicando el método del descenso de gradiente estocástico en el Perceptrón Multicapa se pueden deducir las ecuaciones finales que se utilizan durante el proceso de aprendizaje. Esta deducción lleva detrás un extenso y complejo desarrollo matemático, que resulta de obtener las derivadas del error respecto a los parámetros de la red [1]. Las ecuaciones que se deducen a través de este desarrollo y que constituyen la Regla Delta Generalizada o Algoritmo de RetroPropagación son las siguientes:

- Para actualizar los parámetros de las neuronas de la penúltima capa, $c-1$, a la capa de salida se aplican las siguientes ecuaciones:

    - Para los pesos:

$$w_{ji}^{c-1}(n) = w_{ji}^{c-1}(n-1) + \alpha \delta_i^c a_j^{c-1}(n)$$

Ecuación 10: Ecuación para actualizar los pesos en la capa de salida.

    - Para los Umbrales:

$$u_i^c(n) = u_i^c(n-1) + \alpha \delta_i^c(n)$$

Ecuación 11: Ecuación para actualizar los umbrales en la capa de salida.

donde δ es:

$$\delta_i^c(n) = (s_i - y_i)y_i(1 - y_i)$$

Ecuación 12: Ecuación calcular δ en la capa de salida.

siendo w_{ji}^{c-1} los pesos que van de la neurona j de la capa $c-1$ a la neurona i de la capa c, donde c es la capa de salida y u_i^c el umbral de la neurona i de la última capa.

- Para actualizar los parámetros de las demás capas se aplican las siguientes ecuaciones:
  - Para los pesos:

$$w_{kj}^c(n) = w_{kj}^c(n-1) + \alpha\delta_j^{c+1}a_k^c(n)$$

Ecuación 13: Ecuación para actualizar los pesos en las capas ocultas y de entrada.

  - Para los Umbrales:

$$u_j^{c+1}(n) = u_j^{c+1}(n-1) + \alpha\delta_j^{c+1}(n)$$

Ecuación 14: Ecuación para actualizar los umbrales en las capas ocultas y de entrada.

donde δ es:

$$\delta_j^{c+1} = a_j^{c+1}(1 - a_j^{c+1})\sum_{i=1}^{n_{c+2}} \delta_i^{c+2}\, w_{ji}^{c+1}$$

Ecuación 15: Ecuación para calcular δ en las capas ocultas y de entrada.

donde w_{kj}^c es el peso que une la neurona k de la capa c a la neurona j de la capa $c+1$ y u_j^{c+1} el umbral de la neurona j en la capa $c+1$.

Estas ecuaciones se fundamentan en el siguiente principio, dado un peso que une dos neuronas, una de salida y una de llegada, el peso se actualiza en función del δde la neurona de llegada y de la activación a de la neurona de salida.

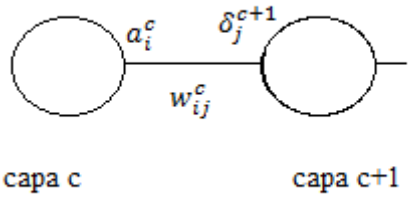

Ilustración 6: Estructura de un peso w a actualizar.

Cuando la neurona de llegada pertenece a la capa de salida δ es el error cometido $(s_i - y_i)$ multiplicado por la derivada de la función de dicha neurona $(y_i * (1 - y_i))$, mientras que cuando la neurona de llegada del peso pertenece a una capa oculta, δ es la derivada de la función de la neurona de llegada $(a_j^{c+1} * (1 - a_i^{c+1}))$ multiplicada por el sumatorio de los pesos que salen de ésta multiplicados por los deltas de las neuronas de llegada $(\sum_{i=1}^{n_{c+2}} \delta_i^{c+2} w_{ji}^{c+1})$.

La Regla Delta Generalizada o Algoritmo de RetroPropagación hace uso de estas ecuaciones para actualizar los distintos parámetros de la red para minimizar así el error E (ecuación 7). Estas ecuaciones se aplican durante el proceso de aprendizaje o entrenamiento de la red. Dicho proceso funciona de la siguiente manera:

1. Se inicializan aleatoriamente los pesos y umbrales de la red con valores próximos a 0.
2. Se presenta un patrón de entrenamiento n a la red y se propaga hacia la salida obteniendo así la respuesta de la red para dicho patrón.
3. Se calcula el error cuadrático $\mathrm{e(n)}$ (ecuación 8) para dicho patrón.
4. Se aplica la Regla Delta Generalizada para modificar los parámetros de la red.
    4.1. Se calculan los valores δ para las neuronas de la capa de salida.
    4.2. Se calculan los valores δ para las neuronas de las capas ocultas, comenzando por la capa contigua a la capa de salida.
    4.3. Se modifican los pesos y los umbrales.
5. Se repiten los pasos 2, 3 y 4 hasta que se hayan presentado todos los patrones de entrenamiento, completando así un ciclo de aprendizaje.
6. Se evalúa el error medio E (ecuación 7) cometido por la red sobre los patrones de entrenamiento y sobre los patrones de test.
7. Se repiten los pasos 2, 3, 4, 5 y 6 hasta que se satisfaga el criterio de parada seleccionado para el entrenamiento. Se denomina criterio de parada a una restricción o requisito que se ha de satisfacer para poder finalizar el proceso de entrenamiento. Los criterios de parada más comunes en el Perceptrón Multicapa son:
    7.1. Haber realizado el número de iteraciones o ciclos de aprendizaje que se haya definido previamente.
    7.2. Un aumento en el error E (ecuación 7) sobre los datos de Test.
    7.3. El error E (ecuación 7) sobre los patrones de entrenamiento no varía durante repetidas iteraciones.
    7.4. Una combinación de los anteriores.

2.3. Ventajas e inconvenientes

El Perceptrón Multicapa es una de las RNAs que más se utiliza en la actualidad debido fundamentalmente a su capacidad como aproximador universal y a su popularidad. Esto no implica que el Perceptrón Multicapa sea la mejor RNA para cualquier tipo de problemas, si no que, al igual que otras arquitecturas de RNAs, el Perceptrón Multicapa cuenta con una serie de ventajas y de deficiencias.

Es importante conocer y sopesar estas ventajas e inconvenientes antes de seleccionar el Perceptrón Multicapa como la herramienta que se va a utilizar para la resolución de un problema.

A continuación se explican las diferentes ventajas e inconvenientes que conlleva el uso del Perceptrón Multicapa:

- Las siguientes características son algunas de las ventajas más importantes que ofrece el Perceptrón Multicapa:

    - Resolución de problemas no lineales: debido al uso de funciones de activación no lineales el Perceptrón Multicapa es capaz de, a partir de un conjunto de ejemplos, aproximar relaciones no lineales.
    - Es un aproximador universal: el Perceptrón Multicapa tiene la capacidad de aproximar cualquier función continua sobre un compacto de R^n.
    - Filtran ruido: el Perceptrón Multicapa tiene una gran capacidad de filtrar el ruido en los datos de entrada, lo cual significa que tiene la habilidad de responder correctamente cuando se le presenta una entrada incompleta o con ruido
    - Gran capacidad de aprendizaje: el Perceptrón Multicapa tiene la capacidad de llevar a cabo el proceso de entrenamiento y aprendizaje a partir de conjuntos reducidos de datos obteniendo buenos resultados, aunque sus resultados mejoran con un mayor conjunto de datos.
    - Independencia del contexto de los datos: debido a que las RNAs interpretan los datos únicamente como valores numéricos, son capaces de resolver numerosos problemas en diferentes campos de conocimiento.
    - Sencillo: Su utilización es sencilla y no requiere de grandes conocimientos previos en la materia.

- Las siguientes características son algunas de las deficiencias más importantes del Perceptrón Multicapa:

    - Largo proceso de entrenamiento: el proceso de entrenamiento del Perceptrón Multicapa puede llegar a ser muy largo cuando se trata de resolver problemas complejos que dependen de un gran número de variables.

- Influencia de la codificación: el Perceptrón Multicapa como RNA únicamente está diseñada para interpretar variables numéricas. Esto implica que en aquellos problemas que contengan variables que no sean numéricas será necesario realizar una codificación de las mismas, pero el resultado de la red depende en gran medida de la codificación elegida.
- Los mínimos locales: el objetivo del Perceptrón Multicapa como se ha explicado anteriormente es la minimización de la función de error E (ecuación 7). La búsqueda de este mínimo implica que el proceso de aprendizaje modifica los parámetros de la red recorriendo así la superficie que define el error E (ecuación 7), la cuál es compleja y se encuentra llena de valles y colinas. El uso del método del descenso del gradiente para la búsqueda de un mínimo en esta función de error implica el riesgo de que se finalice en un mínimo local en lugar del mínimo global que es el objetivo.
- El fenómeno de la parálisis también conocido como "saturación": este fenómeno se debe a que las funciones de activación de las neuronas poseen dos asíntotas horizontales (ilustración 3 e ilustración 4). Debido a esto cuando los valores de entrada a una neurona son muy altos, tanto positivos como negativos, la neurona se satura y su salida siempre toma su valor máximo o mínimo, 0 y 1 en el caso de la función sigmoidal (ilustración 3) y -1 y 1 en el caso de la función hiperbólica (ilustración 4). Esto hace que el aprendizaje se estanque y no avance, pudiendo dar la sensación que el aprendizaje ha finalizado en un mínimo local. La saturación de las neuronas puede evitarse normalizando las variables de entrada y salida de la red.

Capítulo 3. Análisis del sistema

En el presente capítulo se realizará un análisis en profundidad del sistema que se va a desarrollar con la finalidad de identificar los objetivos que ha de satisfacer el simulador para que funcione correctamente y conseguir la conformidad de sus usuarios. De esta manera se determinarán los distintos detalles en los que se basará el desarrollo del simulador.

3.1. Definición de los requisitos del sistema

A continuación se detallarán los requisitos de usuario que se han determinado para el simulador. Estos requisitos recogerán las distintas capacidades y restricciones del mismo, en los denominados requisitos de capacidad y de restricción, respectivamente. Los requisitos de usuario, por lo tanto, reflejan los servicios del sistema y las restricciones bajo las que debe operar desde el punto de vista del usuario. Más adelante se especificarán los requisitos de Software que basándose en los requisitos de usuario determinan las descripciones detalladas de los servicios de sistema desde el punto de vista del desarrollador.

Para facilitar la comprensión de los requisitos y su claridad, éstos se presentan a continuación en una serie de tablas, donde cada requisito se corresponde con una tabla, donde se detalla la siguiente información acerca de cada requisito:

- Identificador unívoco del requisito. Este identificador seguirá la siguiente nomenclatura:
  - Para los requisitos de capacidad: “RUC-XX” donde RUC significa Requisito de Usuario de Capacidad y XX es un identificador numérico de dos dígitos.
  - Para los requisitos de restricción: “RUR-XX” donde RUR significa Requisito de Usuario de Restricción y XX es un identificador numérico de dos dígitos.
- El tipo de requisito. Si se trata de un requisito de capacidad o de restricción.
- La prioridad con la que se caracteriza el requisito.
- La opcionalidad o necesidad de que el requisito esté presente en el simulador final.
- Breve descripción del mismo.

3.1.1. Requisitos de usuario de capacidad

A lo largo de esta sección se presentan los requisitos de capacidad del sistema, los cuales recogen todas aquellas funcionalidades o capacidades que el usuario considera que debe ofrecer el sistema.

Identificador	RUC-01					
Nombre	Definición de redes.					
Tipo	**Capacidad**		X	**Restricción**		
Prioridad	**Alta**	X	**Media**		**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrá definir una arquitectura de red.					

Tabla 1: Requisito de Usuario de Capacidad 01.

Identificador	RUC-02					
Nombre	Modificación de redes.					
Tipo	**Capacidad**		X	**Restricción**		
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrá modificar una arquitectura de red.					

Tabla 2: Requisito de Usuario de Capacidad 02.

Identificador	RUC-03					
Nombre	Carga de patrones de datos.					
Tipo	**Capacidad**		X	**Restricción**		
Prioridad	**Alta**	X	**Media**		**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrá cargar ficheros con los patrones de datos para su uso en los procesos de entrenamiento y test.					

Tabla 3: Requisito de Usuario de Capacidad 03.

Identificador	RUC-04					
Nombre	Selección de la función de activación de las neuronas de salida.					
Tipo	**Capacidad**		X	**Restricción**		
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**			**Opcional**		X
Descripción	Se podrá seleccionar la función de activación de las neuronas de la capa de salida entre función sigmoidal y lineal.					

Tabla 4: Requisito de Usuario de Capacidad 04.

Identificador	RUC-05					
Nombre	Definición de los parámetros de ejecución del entrenamiento.					
Tipo	**Capacidad**		X	**Restricción**		
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrá definir los parámetros de ejecución del entrenamiento, tales como la tasa de aprendizaje, el número de iteraciones y el intervalo entre comprobaciones en la evolución del error.					

Tabla 5: Requisito de Usuario de Capacidad 05.

Identificador	RUC-06					
Nombre	Inicialización de los pesos.					
Tipo	**Capacidad**		X	**Restricción**		
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**			**Opcional**		X
Descripción	Se podrá definir inicializar los pesos aleatoriamente para el entrenamiento, o en caso de que ya tengan valores asignados, continuar utilizándolos.					

Tabla 6: Requisito de Usuario de Capacidad 06.

Identificador	RUC-07					
Nombre	Entrenamiento.					
Tipo	**Capacidad**		X	**Restricción**		
Prioridad	**Alta**	X	**Media**		**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrá realizar entrenamientos de de redes.					

Tabla 7: Requisito de Usuario de Capacidad 07.

Identificador	RUC-08					
Nombre	Calculo del error.					
Tipo	**Capacidad**		X	**Restricción**		
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**			**Opcional**		X
Descripción	Se podrá calcular el error de la arquitectura actual sobre los ficheros de entrenamiento y test.					

Tabla 8: Requisito de Usuario de Capacidad 08.

Identificador	RUC-09					
Nombre	Almacenar los resultados.					
Tipo	Capacidad		X	Restricción		
Prioridad	Alta	X	Media		Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá almacenar los resultados generados por la red para los patrones de entrenamiento y test.					

Tabla 9: Requisito de Usuario de Capacidad 09.

Identificador	RUC-10					
Nombre	Almacenar la red.					
Tipo	Capacidad		X	Restricción		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá almacenar una red y sus parámetros.					

Tabla 10: Requisito de Usuario de Capacidad 10.

Identificador	RUC-11					
Nombre	Cargar una red.					
Tipo	Capacidad		X	Restricción		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá cargar redes almacenadas previamente para continuar utilizándolas.					

Tabla 11: Requisito de Usuario de Capacidad 11.

3.1.2. Requisitos de usuario de restricción

A lo largo de esta sección se presentan los requisitos de restricción del sistema, los cuales recogen las restricciones que se deben respetar durante el desarrollo del simulador.

Identificador	RUR-01					
Nombre	Idioma del simulador.					
Tipo	Capacidad			Restricción		X
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Toda la información de la aplicación y su contenido visual estará en castellano.					

Tabla 12: Requisito de Usuario de Restricción 01.

Identificador	RUR-02					
Nombre	Mensajes de error.					
Tipo	**Capacidad**			**Restricción**		X
Prioridad	**Alta**	X	**Media**		**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Cuando se introduzcan parámetros incorrectos o la herramienta detecte un error, devolverá mensajes de ayuda y de error indicando el problema.					

Tabla 13: Requisito de Usuario de Restricción 02.

Identificador	RUR-03					
Nombre	Portabilidad del sistema.					
Tipo	**Capacidad**			**Restricción**		X
Prioridad	**Alta**	X	**Media**		**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	El sistema será portable entre equipos y no requerirá de instalación para su utilización.					

Tabla 14: Requisito de Usuario de Restricción 03.

Identificador	RUR-04					
Nombre	Formato del almacenamiento de resultados.					
Tipo	**Capacidad**			**Restricción**		X
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**			**Opcional**		X
Descripción	Los resultados se guardarán en un formato sencillo, comprensible y fácilmente exportable a otras herramientas como Excel.					

Tabla 15: Requisito de usuario de Restricción 04.

Identificador	RUR-05					
Nombre	Formato del almacenamiento de red.					
Tipo	**Capacidad**			**Restricción**		X
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**			**Opcional**		X
Descripción	Las redes se guardarán en un formato sencillo y comprensible para su utilización por parte del usuario.					

Tabla 16: Requisito de Usuario de Restricción 05.

Identificador	RUR-06					
Nombre	Visualización de la evolución del error.					
Tipo	Capacidad			Restricción		X
Prioridad	Alta		Media		Baja	X
Opcionalidad	Obligatorio			Opcional		X
Descripción	Los errores sobre los patrones de entrenamiento y test deberán poder visualizarse de manera gráfica.					

Tabla 17: Requisito de Usuario de Restricción 06.

Identificador	RUR-07					
Nombre	Limitación de las funcionalidades.					
Tipo	Capacidad			Restricción		X
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Las funcionalidades que no deban utilizarse en un determinado momento deberán estar deshabilitadas.					

Tabla 18: Requisito de Usuario de Restricción 07.

3.2. Estudio de las alternativas de solución

Una vez analizadas y definidas las necesidades del sistema y los requisitos del usuario es necesario definir una solución tecnológica para poder comenzar con el desarrollo del simulador. Para ello es necesario definir y analizar diferentes alternativas de solución para poder seleccionar aquélla que se adapte mejor a las necesidades del sistema.

Las alternativas de solución analizan el sistema que se va a desarrollar y en función de ello definen las características del entorno en el que se va a llevar a cabo el desarrollo y las herramientas que se van a utilizar para dicho propósito.

Tras analizar numerosas alternativas de solución, la lista de opciones se redujo a la siguiente:

- Alternativa de solución 1:

Sistema Operativo	Ubuntu Linux
Lenguaje de programación	C++
Entorno de desarrollo	Eclipse

Tabla 19: Alternativa de solución 1.

- Alternativa de solución 2:

Sistema Operativo	Microsoft Windows 7
Lenguaje de programación	Java
Entorno de desarrollo	Eclipse

Tabla 20: Alternativa de solución 2.

- Alternativa de solución 3:

Sistema Operativo	Microsoft Windows 7
Lenguaje de programación	Java
Entorno de desarrollo	NetBeans

Tabla 21: Alternativa de solución 3.

- Alternativa de solución4:

Sistema Operativo	Microsoft Windows 7
Lenguaje de programación	C#
Entorno de desarrollo	NetBeans

Tabla 22: Alternativa de solución 4.

Estas alternativas se seleccionaron como las más viables entre las que se estudiaron. Los criterios que se siguieron para seleccionar unas opciones frente a otras fueron sencillos:

- En primer lugar se descartaron aquellas alternativas que se basaban en lenguajes de programación sobre los que el desarrollador no tenía conocimientos previos para facilitar así la fase de adquisición de conocimiento, lo cual redujo las opciones a aquellas basadas en Java, C++ y C#.
- En segundo lugar se descartaron aquellas alternativas que se basaban en entornos o herramientas de desarrollo que requerían la compra de una licencia, lo cual incrementaría ampliamente el presupuesto del proyecto. Esto redujo las opciones a aquellas basadas en los entornos de Eclipse y NetBeans.
- Por último se seleccionó entre las opciones restantes aquellas que tenían un Sistema Operativo de fácil acceso, es decir, que sea sencillo encontrar equipos que tengan dicho Sistema Operativo para poder trabajar sin necesidad de llevar el equipo personal del desarrollador, lo cual redujo las opciones a aquellas basadas en Microsoft Windows 7 y Ubuntu.

Una vez reducida la lista a estas opciones, se estudió en profundidad cada una de ellas, tras lo cual se optó finalmente por la alternativa 2 (tabla 20). Tras este estudio se comprobó que todas las opciones eran viables, por lo cual los motivos para seleccionar esta alternativa frente a las demás, aparte de cumplir con los criterios previamente mencionados, fueron los siguientes:

- La mayor portabilidad que ofrece Java frente a C++ y C#-
- Una mayor experiencia del desarrollador utilizando Java frente a otros lenguajes.
- La mayor accesibilidad a equipos con Microsoft Windows 7.
- Una mayor experiencia del desarrollador utilizando Eclipse frente a NetBeans.

3.3. Requisitos de software y casos de uso

En el presente apartado se detallarán los casos de uso del sistema, los cuales muestran las interacciones del usuario con el mismo, facilitando así una representación gráfica de la relación Usuario-Sistema. Los casos de uso facilitan una interpretación de las necesidades del sistema.

Una vez detallados y explicados los diferentes casos de uso concebidos para el sistema, se detallarán los requisitos de software que identifican las diferentes funcionalidades que se infieren a partir de los requisitos de usuario previamente definidos y los casos de uso que se definen a continuación.

Para facilitar la comprensión de los requisitos y su claridad, al igual que se hiciese con los requisitos de usuario, éstos se presentan a continuación en una serie de tablas, donde cada requisito se corresponde con una tabla, donde se detalla la siguiente información:

- Identificador unívoco del requisito. Este identificador seguirá la siguiente nomenclatura:
    - Para los requisitos funcionales: “RSF-XX” donde RSF significa Requisito de Software Funcional y XX es un identificador numérico de dos dígitos.
    - Para los requisitos de restricción: “RSNF-XX” donde RSNF significa Requisito de Software No Funcional y XX es un identificador numérico de dos dígitos.
- El tipo de requisito. Si se trata de un requisito de funcional o no funcional.
- La prioridad con la que se caracteriza el requisito.
- La opcionalidad o necesidad de que el requisito esté presente en el simulador final.
- Breve descripción del mismo.

3.3.1. Casos de uso

A continuación se muestra de manera gráfica (ilustración 7) un esquema que recoge los diferentes casos de uso que se han recogido para el sistema, en el que se detallan las distintas interacciones del usuario con el sistema.

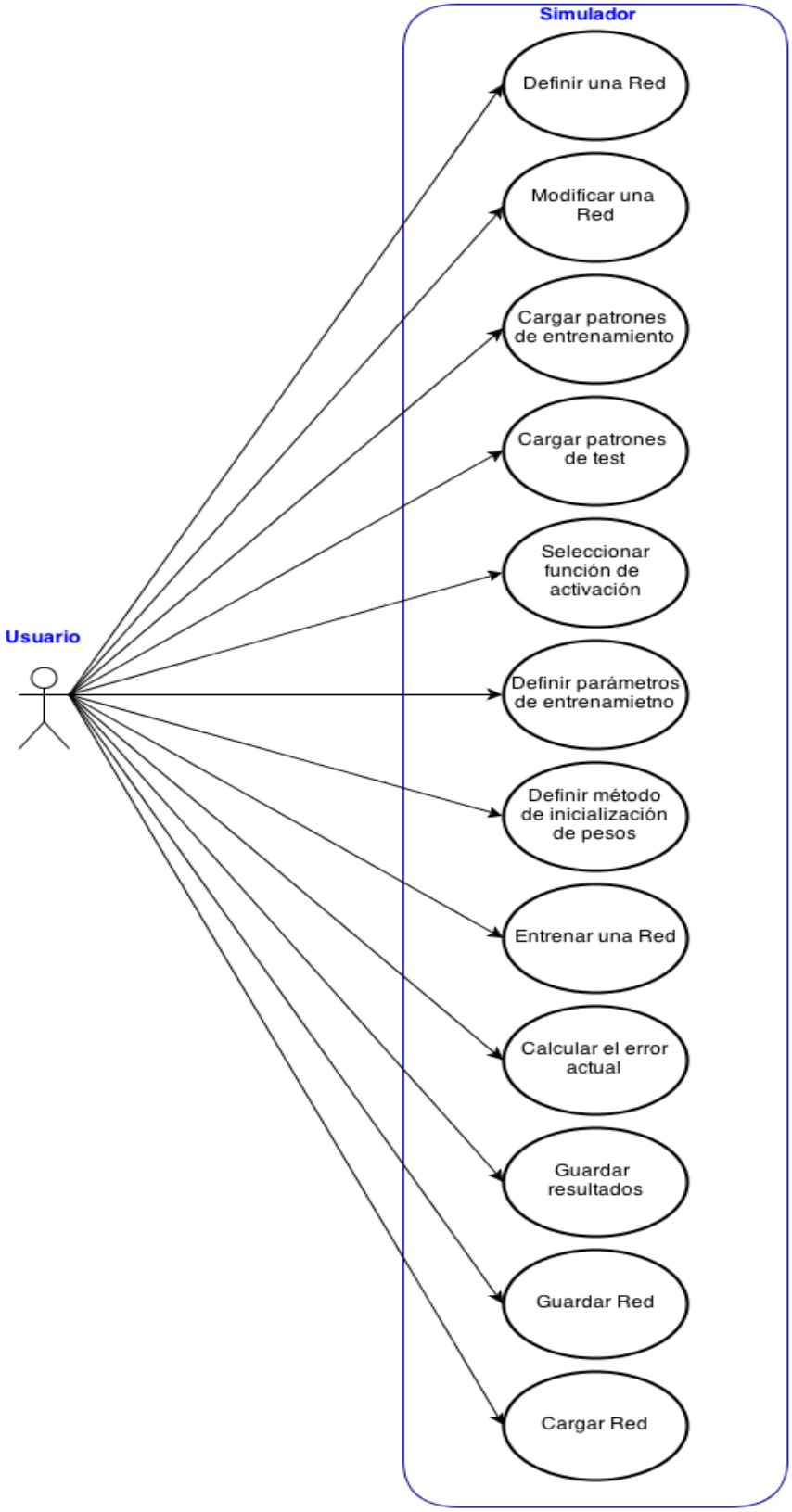

Ilustración 7: Esquema de casos de uso del sistema.

A continuación se detallará en mayor profundidad cada uno de los casos de uso que aparecen en el gráfico anterior (ilustración 7):

Caso de uso	Definir una red
Actores	Usuario
Objetivo	Definir una arquitectura de red para el Perceptrón Multicapa.
Escenario inicial	El simulador se acaba de iniciar.
Escenario final	Se define la red con el número de entradas, el número de salidas y el número de capas ocultas con las neuronas que contiene cada una.

Tabla 23: Caso de uso I (Definir una Red).

Caso de uso	Modificar una red
Actores	Usuario
Objetivo	Modificar una red definida previamente.
Escenario inicial	El simulador tiene una red definida.
Escenario final	Se modifica la red actual modificando uno o varios de sus parámetros, como el número de entradas, el número de salidas, el número de capas ocultas o las neuronas que contiene cada una.

Tabla 24: Caso de uso II (Modificar una Red).

Caso de uso	Cargar patrones de entrenamiento
Actores	Usuario
Objetivo	Cargar en el sistema los patrones de entrenamiento.
Escenario inicial	Se ha definido una red y se pueden haber cargado previamente o no los patrones de entrenamiento.
Escenario final	Se carga un fichero con los patrones de entrenamiento, si había un fichero anterior, se descarta.

Tabla 25: Caso de uso III (Cargar patrones de entrenamiento).

Caso de uso	Cargar patrones de test
Actores	Usuario
Objetivo	Cargar en el sistema los patrones de test.
Escenario inicial	Se ha definido una red, se ha cargado un fichero de datos de entrenamiento y se pueden haber cargado previamente o no los patrones de test.
Escenario final	Se carga un fichero con los patrones de test, si había un fichero anterior, se descarta.

Tabla 26: Caso de uso IV (Cargar patrones de test).

Caso de uso	Seleccionar función de activación
Actores	Usuario
Objetivo	Seleccionar la función de activación en las neuronas de salida.
Escenario inicial	Se ha definido una arquitectura de red.
Escenario final	Se ha seleccionado una función de activación específica para las neuronas de la capa de salida.

Tabla 27: Caso de uso V (Seleccionar función de activación).

Caso de uso	Definir parámetros de entrenamiento
Actores	Usuario
Objetivo	Definir el número de iteraciones, la tasa de aprendizaje y cada cuántas iteraciones se comprueba la evolución del error.
Escenario inicial	Se ha definido una red y cargado los ficheros de datos de entrenamiento y test.
Escenario final	Se han definido los parámetros necesarios para la ejecución del entrenamiento (iteraciones, intervalo entre comprobaciones de la evolución del error y la tasa de aprendizaje).

Tabla 28: Caso de uso VI (Definir parámetros de entrenamiento).

Caso de uso	Definir métodos de inicialización de los pesos
Actores	Usuario
Objetivo	Definir el método de inicialización de pesos.
Escenario inicial	Se ha definido una red y entrenado la misma al menos una vez.
Escenario final	Se define el método según el cual se inicializarán los pesos la próxima vez que se entrene la red actual.

Tabla 29: Caso de uso VII (Definir método de inicialización de los pesos).

Caso de uso	Entrenar una red
Actores	Usuario
Objetivo	Realizar un proceso de entrenamiento sobre una red previamente definida utilizando los ficheros de datos previamente definidos.
Escenario inicial	Se ha definido una red, se han cargado los ficheros de entrenamiento y test y se han definido los parámetros necesarios para la ejecución del entrenamiento.
Escenario final	Se obtiene una red entrenada, una gráfica con la evolución del error durante el proceso de entrenamiento y el valor del error actual de la red.

Tabla 30: Caso de uso VIII (Entrenar una red).

Caso de uso	Calcular el error actual
Actores	Usuario
Objetivo	Consultar el error actual sobre el fichero de datos deseado.
Escenario inicial	Se ha definido una red y entrenado la misma al menos una vez.
Escenario final	Se le muestra al usuario el error actual sobre el fichero de datos que desee.

Tabla 31: Caso de uso IX (Calcular el error actual).

Caso de uso	Guardar Resultados
Actores	Usuario
Objetivo	Guardar los resultados obtenidos por la red.
Escenario inicial	Se ha definido y entrenado una red y se han cargado los ficheros de datos de entrenamiento y test.
Escenario final	Se genera un fichero de texto con los resultados obtenidos por la red.

Tabla 32: Caso de uso X (Guardar resultados).

Caso de uso	Guardar una red
Actores	Usuario
Objetivo	Guardar una arquitectura de red con toda su información (pesos y umbrales).
Escenario inicial	Se ha definido una red y entrenado la misma.
Escenario final	Se genera un fichero de texto con la arquitectura de la red y la información de ésta.

Tabla 33: Caso de uso XI (Guardar Red).

Caso de uso	Cargar red
Actores	Usuario
Objetivo	Cargar una red previamente guardada.
Escenario inicial	Se tiene un fichero de texto con una red previamente guardada por el simulador.
Escenario final	Se ha cargado la red y ésta se define automáticamente en el simulador.

Tabla 34: Caso de uso XII (Cargar Red).

3.3.2. Requisitos de software funcionales

A continuación se detallan los requisitos funcionales, los cuales establecen las distintas funcionalidades de las que dispondrá el simulador, estableciendo así el comportamiento del sistema:

Identificador	RSF-01					
Nombre	Definición de redes.					
Tipo	**Funcional**		X	**No Funcional**		
Prioridad	**Alta**	X	**Media**		**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrán definir arquitecturas de red del Perceptrón Multicapa.					

Tabla 35: Requisito de Software Funcional 01.

Identificador	RSF-02					
Nombre	Modificación de redes.					
Tipo	**Funcional**		X	**No Funcional**		
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrán modificar arquitecturas de red previamente definidas.					

Tabla 36: Requisito de Software Funcional 02.

Identificador	RSF-03					
Nombre	Carga de patrones de entrenamiento.					
Tipo	**Funcional**		X	**No Funcional**		
Prioridad	**Alta**	X	**Media**		**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrán cargar ficheros que contengan los patrones de datos para entrenamiento.					

Tabla 37: Requisito de Software Funcional 03.

Identificador	RSF-04					
Nombre	Carga de patrones de datos de test.					
Tipo	**Funcional**		X	**No Funcional**		
Prioridad	**Alta**	X	**Media**		**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Se podrán cargar ficheros que contengan los patrones de datos para test.					

Tabla 38: Requisito de Software Funcional 04.

Identificador	RSF-05					
Nombre	Selección de la función de activación de salida.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio			Opcional		X
Descripción	Se podrá seleccionar la función de activación en las neuronas de la capa de salida entre una función sigmoidal y una función lineal.					

Tabla 39: Requisito de Software Funcional 05.

Identificador	RSF-06					
Nombre	Definición de la tasa de aprendizaje.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá definir la tasa de aprendizaje que se utilizara durante el proceso de entrenamiento.					

Tabla 40: Requisito de Software Funcional 06.

Identificador	RSF-07					
Nombre	Definición del número de iteraciones.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá definir el número de iteraciones que dura el proceso de entrenamiento.					

Tabla 41: Requisito de Software Funcional 07.

Identificador	RSF-08					
Nombre	Definición del intervalo entre comprobaciones de error.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio			Opcional		X
Descripción	Se podrá definir el intervalo de iteraciones que transcurre entre comprobaciones de error en el proceso de entrenamiento.					

Tabla 42: Requisito de Software Funcional 08.

Identificador	RSF-09					
Nombre	Definición del método de inicialización de los pesos.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio			Opcional		X
Descripción	Se podrá seleccionar si los pesos se inicializan de manera aleatoria o se utilizan los valores actuales.					

Tabla 43: Requisito de Software Funcional 09.

Identificador	RSF-10					
Nombre	Entrenamiento de una red.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta	X	Media		Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá entrenar arquitecturas de red.					

Tabla 44: Requisito de Software Funcional 10.

Identificador	RSF-11					
Nombre	Calculo del error actual.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio			Opcional		X
Descripción	Se podrá consultar el error actual de la red sobre los patrones de datos de entrenamiento y test.					

Tabla 45: Requisito de Software Funcional 11.

Identificador	RSF-12					
Nombre	Almacenamiento de los resultados.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta	X	Media		Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá almacenar los resultados obtenidos por la red para los patrones de entrenamiento y test.					

Tabla 46: Requisito de Software Funcional 12.

Identificador	RSF-13					
Nombre	Almacenamiento de la red.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá guardar arquitecturas de red así como sus pesos y umbrales.					

Tabla 47: Requisito de Software Funcional 13.

Identificador	RSF-14					
Nombre	Carga de una red.					
Tipo	Funcional		X	No Funcional		
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Se podrá cargar redes previamente guardadas para su reutilización.					

Tabla 48: Requisito de Software Funcional 14.

3.3.3. Requisitos de software no funcionales

A continuación se detallan los requisitos no funcionales, los cuales especifican criterios que limitan el comportamiento del simulador, pero sin aportar funcionalidad al sistema:

Identificador	RSNF-01					
Nombre	Lenguaje del simulador.					
Tipo	Funcional			No Funcional		X
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Toda la información del simulador estará en castellano.					

Tabla 49: Requisito de Software No Funcional 01.

Identificador	RSNF-02					
Nombre	Mensajes de ayuda y error.					
Tipo	Funcional			No Funcional		X
Prioridad	Alta	X	Media		Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	Cuando se introduzcan parámetros incorrectos o la herramienta detecte un error, devolverá mensajes de ayuda y de error indicando el problema.					

Tabla 50: Requisito de Software No Funcional 02.

Identificador	RSNF-03					
Nombre	Programación					
Tipo	Funcional			No Funcional		X
Prioridad	Alta	X	Media		Baja	
Opcionalidad	Obligatorio		X	Opcional		
Descripción	El lenguaje de programación utilizado para la implementación será Java.					

Tabla 51: Requisito de Software No funcional 03.

Identificador	RSNF-04					
Nombre	Formato de almacenamiento de resultados.					
Tipo	Funcional			No Funcional		X
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio			Opcional		X
Descripción	Los resultados se guardarán en un fichero de texto con un formato sencillo, comprensible y fácilmente exportable a otras herramientas como Excel.					

Tabla 52: Requisito de Software No Funcional 04.

Identificador	RSNF-05					
Nombre	Formato de almacenamiento de la red.					
Tipo	Funcional			No Funcional		X
Prioridad	Alta		Media	X	Baja	
Opcionalidad	Obligatorio			Opcional		X
Descripción	Las redes se guardarán en un fichero de texto con un formato sencillo y comprensible para su utilización por parte del usuario.					

Tabla 53: Requisito de Software No Funcional 05.

Identificador	RSNF-06					
Nombre	Visualización del error.					
Tipo	**Funcional**			**No Funcional**		X
Prioridad	**Alta**		**Media**		**Baja**	X
Opcionalidad	**Obligatorio**			**Opcional**		X
Descripción	Los errores sobre los patrones de entrenamiento y test deberán visualizarse de manera gráfica.					

Tabla 54: Requisito de Software No Funcional 06.

Identificador	RSNF-07					
Nombre	Limitación de funcionalidades.					
Tipo	**Funcional**			**No Funcional**		X
Prioridad	**Alta**		**Media**	X	**Baja**	
Opcionalidad	**Obligatorio**		X	**Opcional**		
Descripción	Las funcionalidades que no deban utilizarse estarán deshabilitadas.					

Tabla 55: Requisito de Software No Funcional 07.

3.4. Matriz de trazabilidad

A continuación, mediante las siguientes matrices de trazabilidad, se representan las relaciones entre los requisitos de usuario y de software, de esta manera se puede observar si alguna de las necesidades del usuario no queda representada en las necesidades del sistema final.

3.4.1. Trazabilidad entre requisitos de capacidad y funcionales

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUC-07	RUC-08	RUC-09	RUC-10	RUC-11
RSF-01	X										
RSF-02		X									
RSF-03			X								
RSF-04			X								
RSF-05				X							
RSF-06					X						
RSF-07					X						
RSF-08					X						
RSF-09						X					
RSF-10							X				
RSF-11								X			
RSF-12									X		
RSF-13										X	
RSF-14											X

Tabla 56: Matriz de trazabilidad entre requisitos funcionales y de capacidad.

3.4.2. Trazabilidad entre requisitos de restricción y no funcionales

	RUR-01	RUR-02	RUR-03	RUR-04	RUR-05	RUR-06	RUR-07
RSNF-01	x						
RSNF-02		x					
RSNF-03			x				
RSNF-04				x			
RSNF-05					x		
RSNF-06						x	
RSNF-07							x

Tabla 57: Matriz trazabilidad entre requisitos no funcionales y de restricción.

Capítulo 4. Diseño del sistema

A continuación se detallará el diseño en el que se ha basado el desarrollo del simulador. Este diseño especifica los distintos aspectos técnicos del funcionamiento y de la estructura interna del simulador, de manera que se facilite, en un futuro, llevar a cabo modificaciones por parte de personas que no participaron en el desarrollo del mismo.

Para dicho propósito se explicará la arquitectura del sistema, se facilitará un análisis de las distintas clases que lo componen y se detallará la secuencia de iteraciones que sigue en el código cada una de las distintas funcionalidades.

4.1. Definición de la arquitectura del sistema

Para el correcto desarrollo del simulador es necesario definir desde un principio la arquitectura que va a seguir su desarrollo. Para el desarrollo del simulador se decidió definir una arquitectura enfocada a maximizar la sencillez y eficacia del mismo. Esta arquitectura está compuesta de dos capas, una capa de “vista” y otra de “lógica”.

La primera capa o capa de “vista” representa la interfaz del simulador y actúa como intermediario entre el usuario y la segunda capa. Para ello, cuando el usuario solicita alguna información o funcionalidad del simulador, esta primera capa se encarga de realizar las peticiones necesarias a la segunda, para que ésta devuelva los resultados que se esperan en función de la petición. Esta capa se compone de una única clase que se encargará de realizar las peticiones a la segunda capa y de definir la interfaz del simulador.

La función de la segunda capa o capa de “lógica” es la realización de los diferentes cálculos y funciones del simulador a petición de la primera capa, así como la administración de la diferente información de la que el simulador hace uso. Esta capa se compone de diferentes clases y funciones, de modo que la separación por funciones facilite llevar a cabo futuras modificaciones.

4.2. Análisis de las clases

Una vez establecidos los distintos casos de uso, los requisitos que ha de satisfacer el simulador y la arquitectura del mismo, se procede a determinar la estructura de las distintas clases que conformarán el simulador. Para cada clase se mostrará a continuación, sus responsabilidades dentro del simulador, los atributos que contendrán y las distintas operaciones que deberán realizar.

Clase	Proyecto
Responsabilidades	Lanzar y ejecutar el simulador.
Atributos	Un objeto de la clase Interfaces que se utiliza para lanzar la interfaz del simulador.
Operaciones	Main: método que instancia la clase Interfaces e inicia el simulador.

Tabla 58: Clase Proyecto.

Clase	Interfaces
Responsabilidades	Definir la interfaz del simulador y recoger las peticiones del usuario.
Atributos	Un objeto de la clase Arquitectura para establecer y definir la arquitectura de red. Un objeto de la clase Fichero para gestionar la carga de ficheros de datos. Un objeto de la clase Entrenamiento para disponer de las operaciones de entrenamiento. Un objeto de la clase Escritor para gestionar el guardado de redes y resultados del simulador. Un objeto de la clase Cargador para gestionar la carga de redes.
Operaciones	CrearEntrada: establece el número de neuronas de la capa de entrada de la arquitectura. CrearSalida: establece el número de neuronas de la capa de salida de la arquitectura. AnadirCapa: añade una nueva capa oculta a la arquitectura con las neuronas especificadas. NoAnadirMasCapa: establece que no se pueden añadir más capas ocultas a la arquitectura. ModificarEntradas: habilita la modificación de la capa de entrada de la arquitectura. ModificarSalidas: habilita la modificación de la capa de salida de la arquitectura. ModificarIntermedias: habilita la modificación de las capas ocultas de la arquitectura. CargarFicheroEntrenamiento: permite al usuario seleccionar un fichero y cargarlo en el simulador para que actúe como conjunto de

	datos de entrenamiento. CargarFicheroTest: permite al usuario seleccionar un fichero y cargarlo en el simulador para que actúe como conjunto de datos de test. RealizarEntrenamiento: realiza el proceso de entrenamiento de la red. RealizarTest: muestra al usuario el error de la red actual sobre los conjuntos de datos de entrenamiento y test. GuardarRed: almacena en un fichero la información de la red definida en el simulador. CargarModeloRed: carga en el simulador arquitecturas de red previamente definidas y guardadas. GuardarSalidaRed: almacena en un fichero los resultados obtenidos por la red para los conjuntos de datos de entrenamiento y test.

Tabla 59: Clase Interfaces.

Clase	Arquitectura
Responsabilidades	Establecer la arquitectura de la red, almacenando la información propia de ésta.
Atributos	NeuronasEntrada: el número de neuronas de la capa de entrada. NeuronasSalida: el número de neuronas de la capa de salida. NeuronasXcapa <int>: lista dinámica en la que se van añadiendo el número de neuronas que tiene cada capa oculta. NeuronasXcapaFinal[]: lista estática que especifica el número de neuronas de todas las capas de la red.
Operaciones	Get: existe un método "get" para cada uno de los atributos de la clase que devuelve el valor de dicho atributo. Set: existe un método "set" para cada uno de los atributos de la clase para darle un valor a dicho atributo. AnadirCapa: define una nueva capa oculta. IncrementarCapasOcultas: incrementa el contador de capas ocultas. FinalizarOcultas: finaliza la lista dinámica de capas ocultas y define la lista estática con todas las capas de la red. VaciarCapasOcultas: vacía la lista dinámica de las capas ocultas para poder definirla de nuevo.

Tabla 60: Clase Arquitectura.

Clase	Fichero
Responsabilidades	Lectura, carga e interpretación de los ficheros de datos de entrenamiento y test.
Atributos	Atributos: define la cantidad de atributos que contienen los patrones del fichero de datos que se está cargando. Entradas: define el número de atributos que se corresponden con entradas de la red en los patrones de datos. Salidas: define el número de atributos que se corresponden con salidas de la red en los patrones de datos. InstanciasEntrenamiento <double []>: lista dinámica en la que se almacenan los patrones de datos de entrenamiento. IntanciasTest <double []>: lista dinámica en la que se almacenan los patrones de datos de test. CodigoEntrenamiento: atributo que establece el mensaje de error correspondiente en caso de fallo en la carga de un fichero de datos para su uso como conjunto de entrenamiento. CodigoTest: atributo que establece el mensaje de error correspondiente en caso de fallo en la carga de un fichero de datos para su uso como conjunto de test.
Operaciones	Get: existe un método “get” para cada uno de los atributos de la clase que devuelve el valor de dicho atributo. Set: existe un método “set” para cada uno de los atributos de la clase para darle un valor a dicho atributo. IncrementarPatrones: añade un patrón de datos a la lista dinámica de patrones de entrenamiento. IncrementarPatronesTest: añade un patrón de datos a la lista dinámica de patrones de test. LeerFicheroEntrenamiento: interpreta un fichero de patrones de datos y lo carga en el simulador como conjunto de entrenamiento. LeerFicheroTest: interpreta un fichero de

	patrones de datos y lo carga en el simulador como conjunto de test. EmptyListaEntrenamiento: vacía la lista de patrones de entrenamiento. EmptyListaTest: vacía la lista de patrones de test.

Tabla 61: Clase Fichero.

Clase	**Entrenamiento**
Responsabilidades	Se encarga del aprendizaje de la red y del cálculo de errores sobre los patrones de datos.
Atributos	PatronesEntrenamiento[][]: patrones de datos cargados como conjunto de entrenamiento. PatronesTest[][]: patrones de datos cargados como conjunto de test. Iteracion_max: número de iteraciones que durará el proceso de aprendizaje. Comprobar: intervalo que transcurre entre comprobaciones de evolución del error en el proceso de aprendizaje. Tasa: tasa de aprendizaje que se utilizará para actualizar los pesos en el proceso de aprendizaje. NeuronasEntrada: número de neuronas de la capa de entrada. NeuronasSalida: número de neuronas de la capa de salida. NeuronasXcapa[]: lista que define el número de neuronas de todas las capas de la arquitectura de red. ErrorXiteracionEntrenamiento[]: lista de los errores que se calculan en las comprobaciones de evolución del error para el conjunto de entrenamiento. ErrorXiteracionTest[]: lista de los errores que se calculan en las comprobaciones de evolución del error para el conjunto de test. ErrorEntrenamientoFinal: error final de la red

	una vez terminado el proceso de aprendizaje sobre el conjunto de datos de entrenamiento. ErrorTestFinal: error final de la red una vez terminado el proceso de aprendizaje sobre el conjunto de datos de test. PesoUmbral[][]: umbrales de las neuronas de la red. Pesos[][][]: pesos que ponderan las conexiones inter-neuronales de la red. PesoUmbral_new[][]: umbrales calculados en el proceso de aprendizaje para su actualización. Pesos_new[][][]: pesos calculados en el proceso de aprendizaje de la red para su actualización. Activaciones[][]: salidas calculadas para cada una de las neuronas de una arquitectura de red. Deltas[][]: deltas de las diferentes neuronas de la red que serán utilizadas en el proceso de aprendizaje. Un objeto de la clase Graficador para dibujar la gráfica de evolución del error.
Operaciones	Get: existe un método "get" para cada uno de los atributos de la clase que devuelve el valor de dicho atributo. Set: existe un método "set" para cada uno de los atributos de la clase para darle un valor a dicho atributo. IniciaPesos: define e inicializa la estructura de datos que almacena los pesos y umbrales de la red. IniciaActivaciones: define e inicializa la estructura de datos que almacena las activaciones de las distintas neuronas de la red. IniciaDeltas: define e inicializa la estructura de datos que almacena los deltas de las distintas neuronas de la red. IniciaErrorXIt: define e inicializa las estructuras de datos que almacenan la evolución del error para los conjuntos de entrenamiento y test.

	Sigmoide: calcula el resultado de aplicar la función sigmoidal sobre un valor. CalcularError: calcula el error de la red actual sobre el conjunto de datos de entrenamiento. CalcularErrorTest: calcula el error de la red actual sobre el conjunto de datos de test. FinalizarError: calcula el error de una iteración una vez calculado el error individual de cada patrón del conjunto de datos de entrenamiento. FinalizarErrorTest: calcula el error de una iteración una vez calculado el error individual de cada patrón del conjunto de datos de test. CalcularSalidaRed: calcula la salida de la red para los patrones de entrenamiento. CalcularSalidaRedTest: calcula la salida de la red para los patrones de test. CalcularDeltas: calcula los deltas de las neuronas de la red para su utilización en el proceso de aprendizaje. ActualizarPesos: calcula los nuevos pesos y umbrales de la red actualizando los actuales. Entrenar: lleva a cabo el proceso de aprendizaje de la red haciendo una petición a la Graficador para que genere la gráfica de evolución del error. Testear: calcula el error actual de la red sobre los patrones de datos de entrenamiento y test.

Tabla 62: Clase Entrenamiento.

Clase	Graficador
Responsabilidades	Dibujar la gráfica de evolución del error durante el proceso de aprendizaje.
Atributos	Sin atributos.
Operaciones	Graficar: define y dibuja la gráfica de evolución del error.

Tabla 63: Clase Graficador.

Clase	Cargador
Responsabilidades	Se encarga de interpretar y cargar ficheros que contienen arquitecturas de red.
Atributos	NeuronasXcapa[]: lista que almacena la arquitectura de red cargada, almacenando el número de neuronas que forma cada capa. Umbrales[][]: umbrales de las neuronas de la arquitectura de red cargada. Pesos[][][]: pesos que ponderan las conexiones inter-neuronales de la arquitectura de red cargada.
Operaciones	Get: existe un método "get" para cada uno de los atributos de la clase que devuelve el valor de dicho atributo. CargarFicheroRed: interpreta y carga en el simulador la información de una arquitectura de red almacenada en un fichero de texto.

Tabla 64: Clase Cargador.

Clase	Escritor
Responsabilidades	Se encarga de guardar en ficheros de texto los resultados obtenidos por el simulador y de guardar arquitecturas de red.
Atributos	Sin atributos.
Operaciones	EscribirRed: guarda arquitecturas de red en ficheros de texto. EscribirEntren: guarda las salidas deseadas y las salidas obtenidas por la red para el conjunto de datos de entrenamiento. EscribirTesten: guarda las salidas deseadas y las salidas obtenidas por la red para el conjunto de datos de test.

Tabla 65: Clase Escritor.

4.3. Descripción de las iteraciones de las funcionalidades principales

A continuación se detallarán la secuencia de iteraciones que sigue el código para llevar a cabo cada una de las distintas funcionalidades que lo componen. De esta manera se detallará el funcionamiento a nivel interno del simulador para facilitar así su comprensión.

4.3.1. Iniciar el simulador

La siguiente ilustración (ilustración 8) detalla las iteraciones entre clases que se llevan a cabo a nivel interno para la inicialización del simulador:

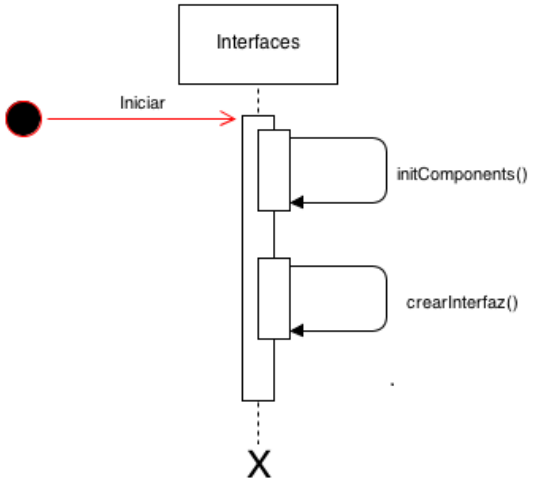

Ilustración 8: Iteraciones en la inicialización del simulador.

4.3.2. Definir una arquitectura

La siguiente ilustración (ilustración 9) detalla las iteraciones entre clases que se llevan a cabo a nivel interno para definir una arquitectura de red, lo cual es un proceso de tres partes que implica la definición de la capa de entrada, de la capa de salida y de las capas ocultas:

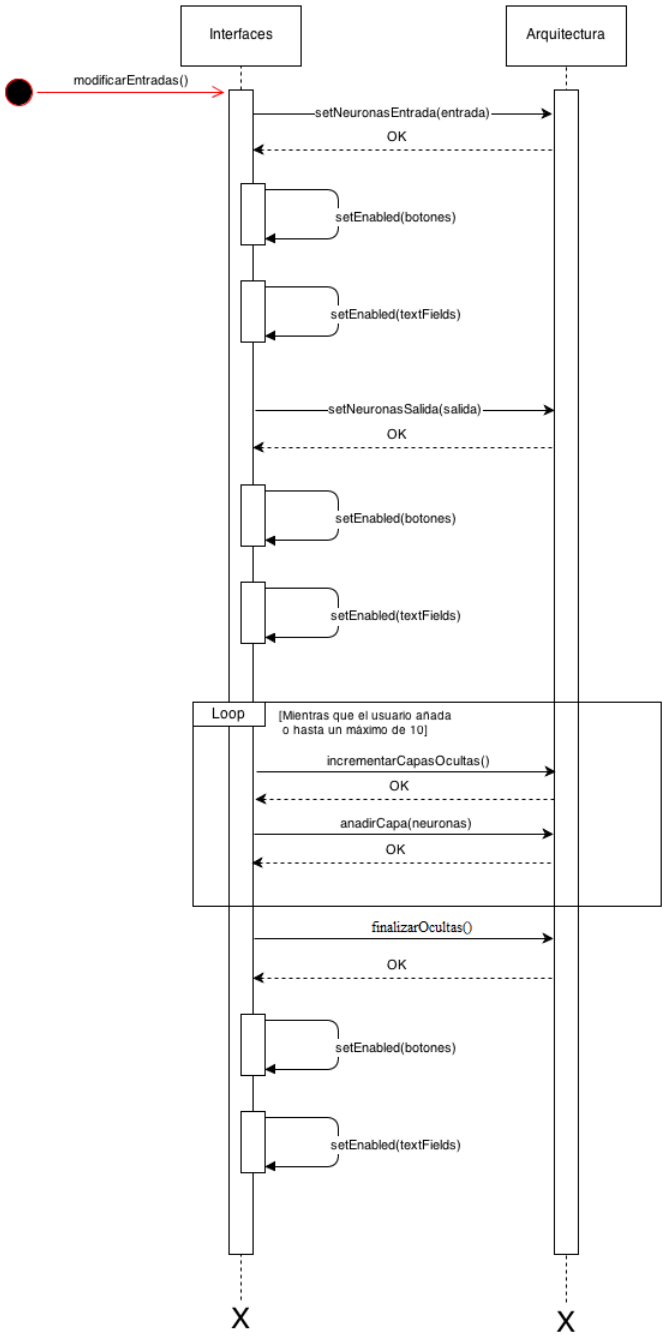

Ilustración 9: Iteraciones en la definición de una arquitectura de red.

4.3.3. Modificar una arquitectura

La siguientes ilustraciones (ilustración 10, ilustración 11 e ilustración 12) detallan las iteraciones entre clases que se llevan a cabo a nivel interno para modificar una arquitectura de red. Estas ilustraciones detallan la modificación de la capa de entrada, de la capa de salida y de las capas ocultas de la arquitectura respectivamente:

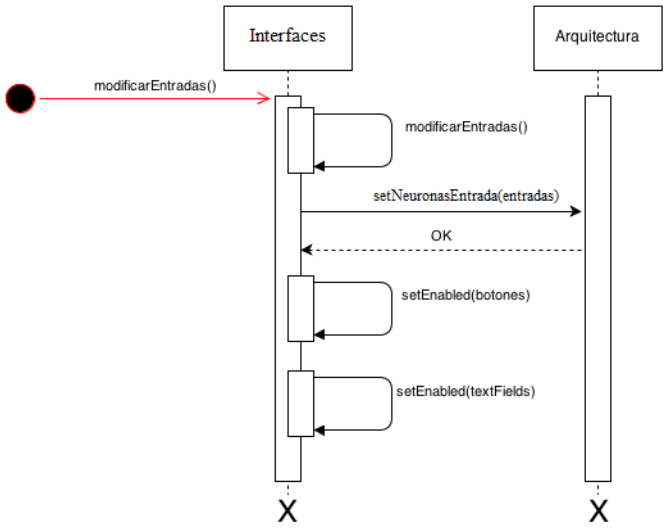

Ilustración 10: Iteraciones para la modificación de la capa de entrada.

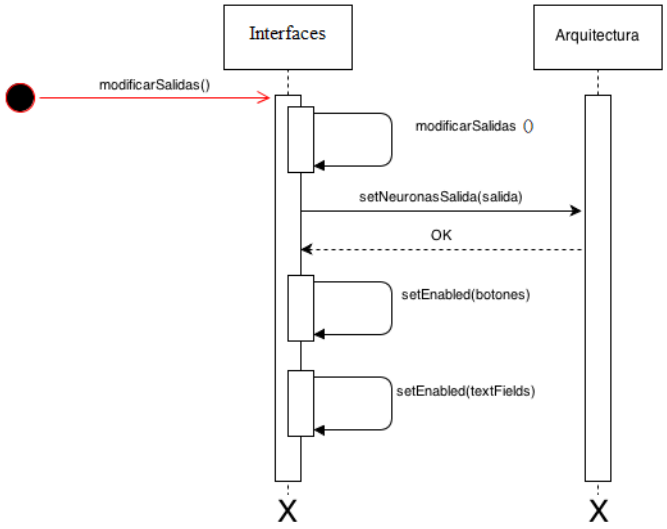

Ilustración 11: Iteraciones para la modificación de la capa de salida.

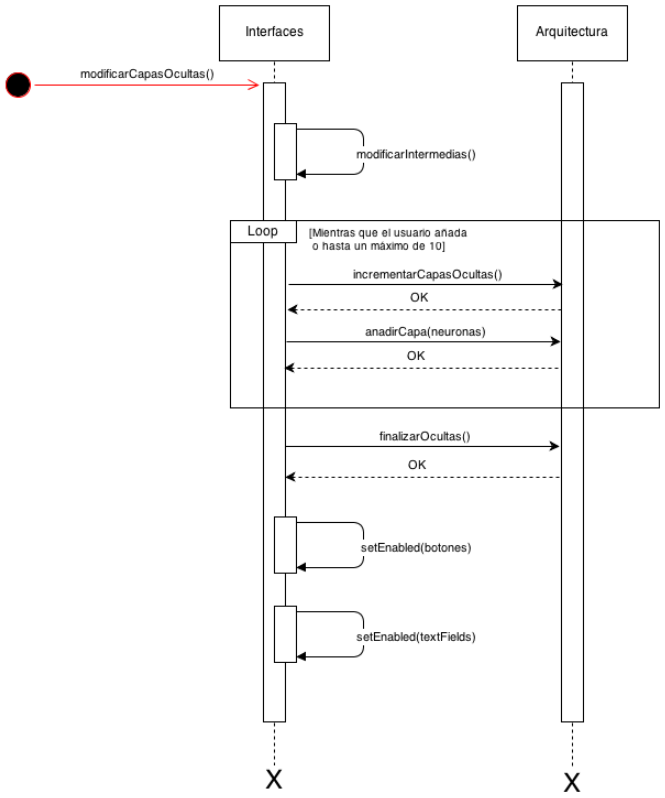

Ilustración 12: Iteraciones para la modificación de las capas ocultas.

4.3.4. Cargar los ficheros de datos

La siguientes ilustraciones (ilustración 13 e ilustración 14) detallan las iteraciones entre clases que se llevan a cabo a nivel interno para cargar ficheros de datos. Estas ilustraciones detallan la carga de ficheros de datos de entrenamiento y test respectivamente:

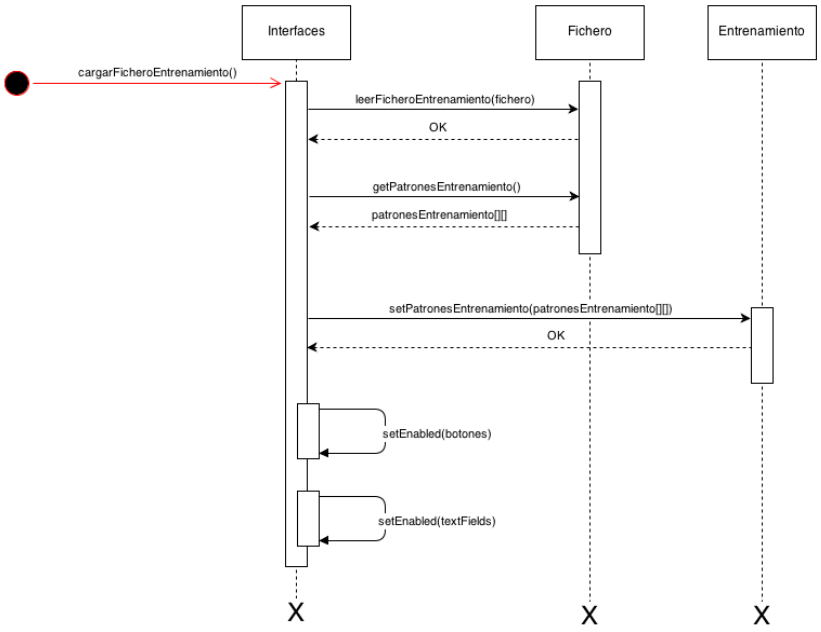

Ilustración 13: Iteraciones para la carga de ficheros de datos de entrenamiento.

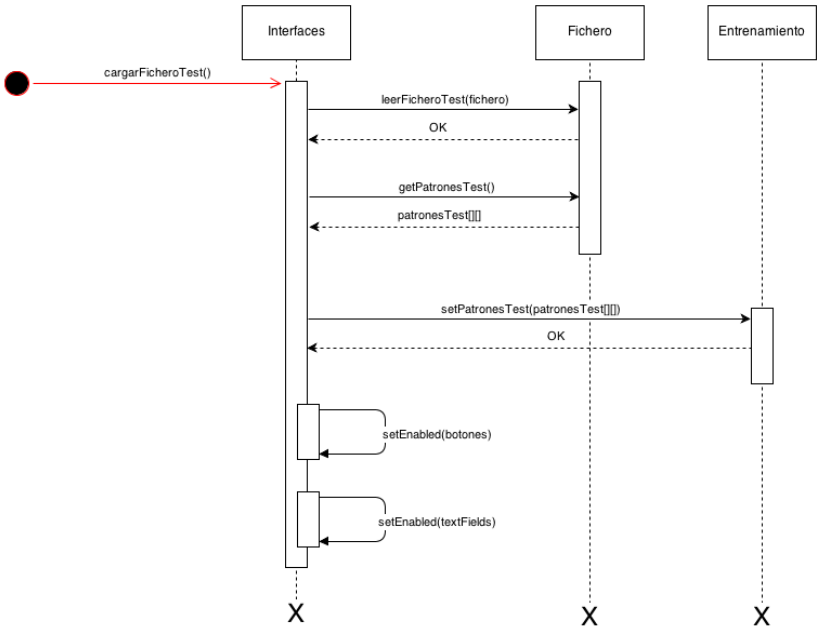

Ilustración 14: Iteraciones para la carga de ficheros de datos de test.

4.3.5. Entrenar una red

La siguiente ilustración (ilustración 15) detalla las iteraciones entre clases que se llevan a cabo a nivel interno para llevar a cabo el proceso de aprendizaje de la red:

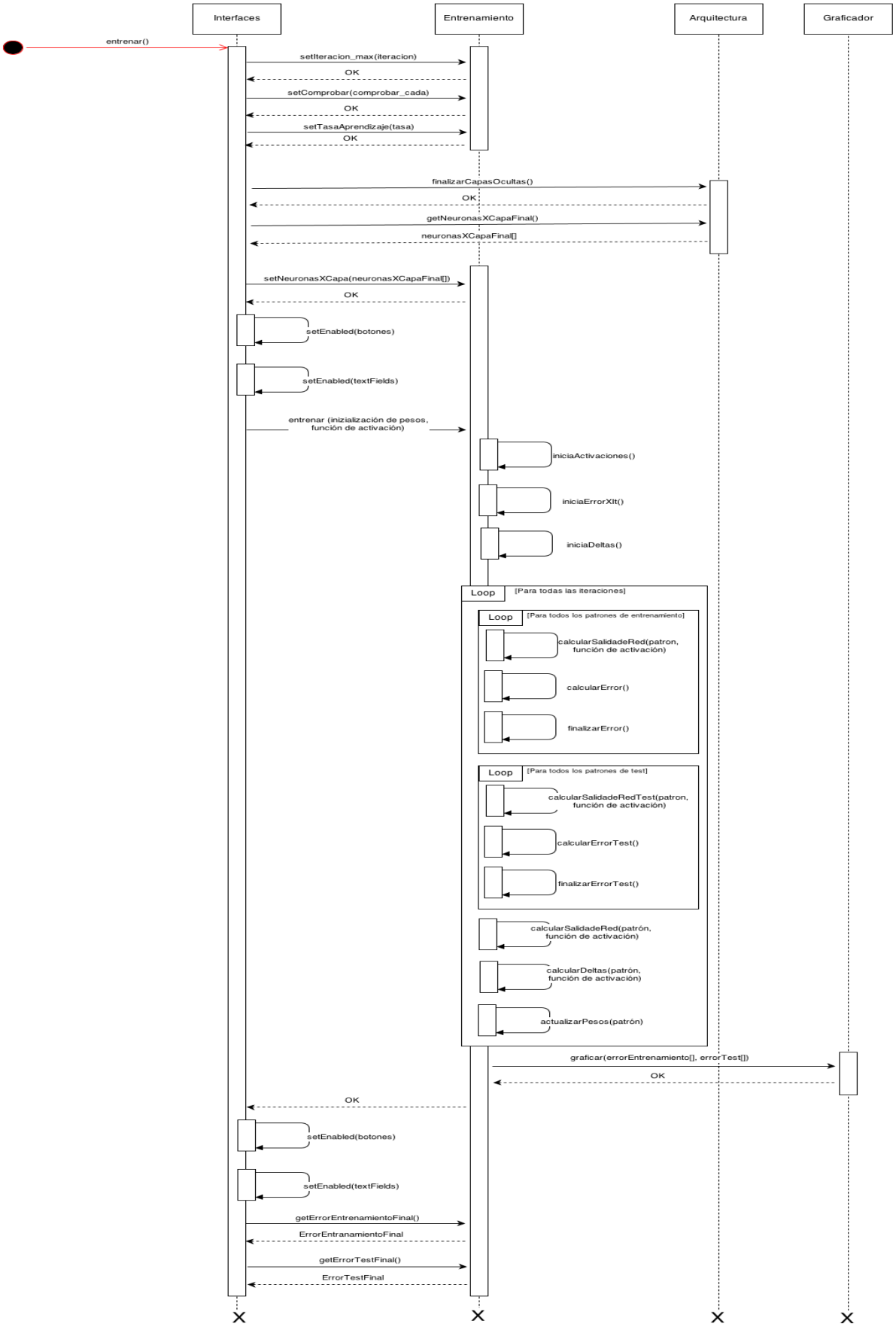

Ilustración 15: Iteraciones para el entrenamiento de una red.

4.3.6. Calculo del error actual

La siguiente ilustración (ilustración 16) detalla las iteraciones entre clases que se llevan a cabo a nivel interno para consultar el error actual de la red sobre los conjuntos de datos de entrenamiento y test:

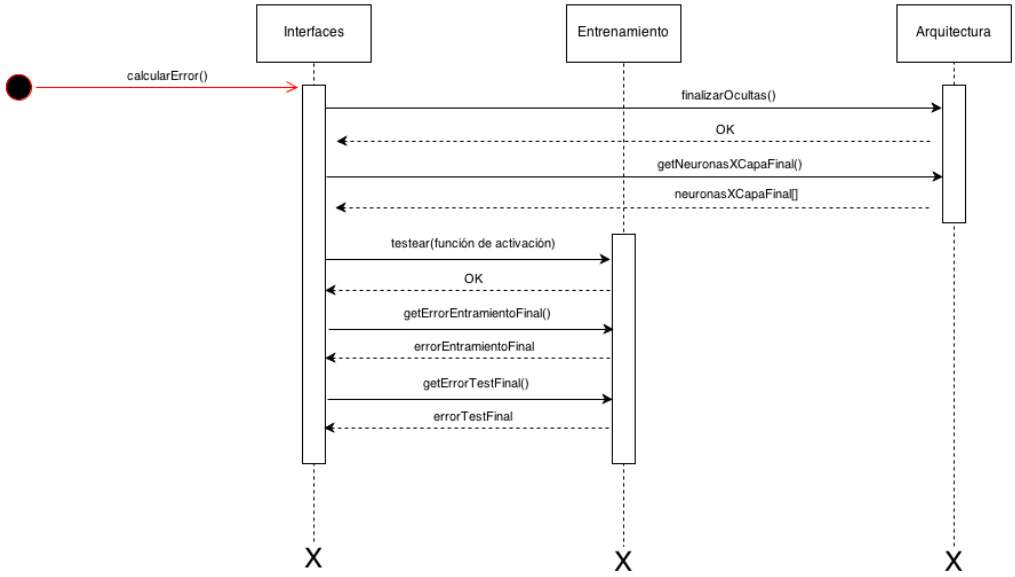

Ilustración 16: Iteraciones para el cálculo del error actual.

4.3.7. Cargar una red

La siguiente ilustración (ilustración 17) detalla las iteraciones entre clases que se llevan a cabo a nivel interno para cargar una arquitectura de red en el simulador:

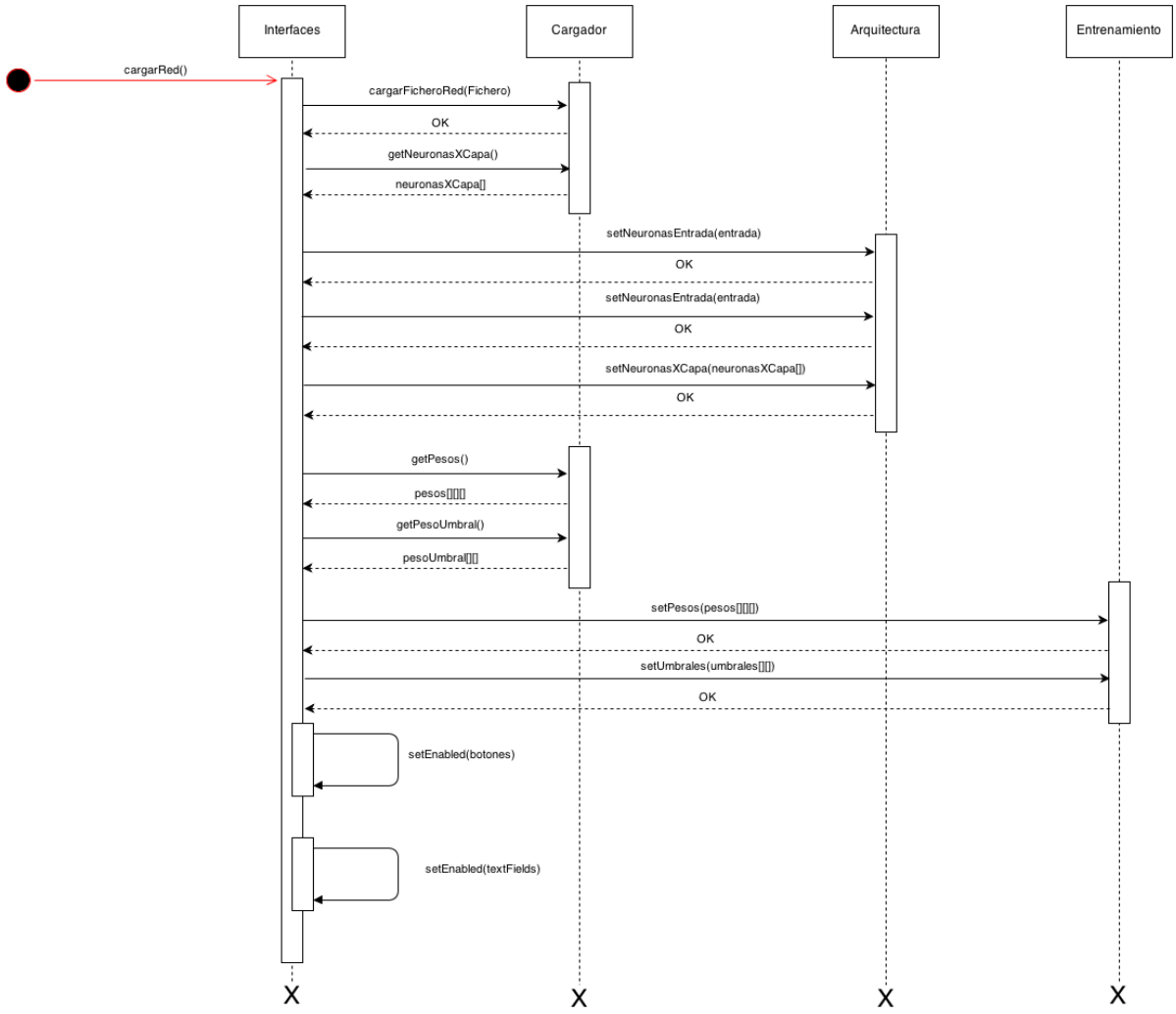

Ilustración 17: Iteraciones para la carga de arquitecturas de red.

4.3.8. Guardar una red

La siguiente ilustración (ilustración 18) detalla las iteraciones entre clases que se llevan a cabo a nivel interno para guardar una arquitectura de red en un fichero de texto:

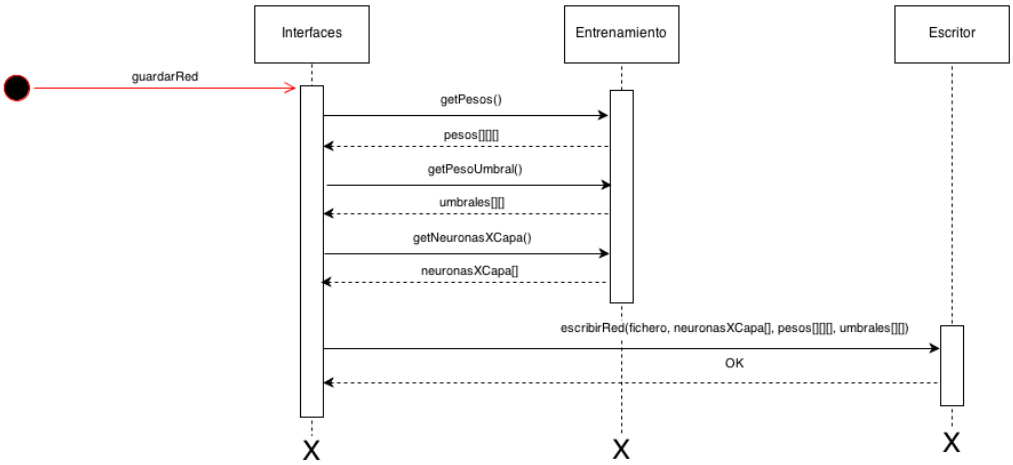

Ilustración 18: Iteraciones para guardar una arquitectura de red.

4.3.9. Guardar los resultados de una red

La siguientes ilustraciones (ilustración 19 e ilustración 20) detallan las iteraciones entre clases que se llevan a cabo a nivel interno para guardar los resultados obtenidos por la arquitectura de red definida en el simulador, para los patrones de los ficheros de datos. Estas ilustraciones detallan el guardado de los resultados obtenidos para los datos de los ficheros de entrenamiento y test, respectivamente:

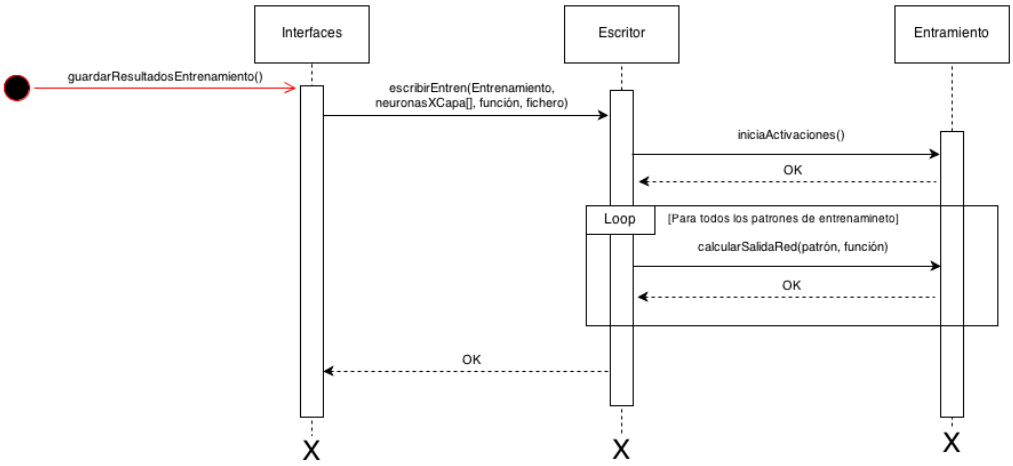

Ilustración 19: Iteraciones para el guardado de resultados para los patrones de entrenamiento.

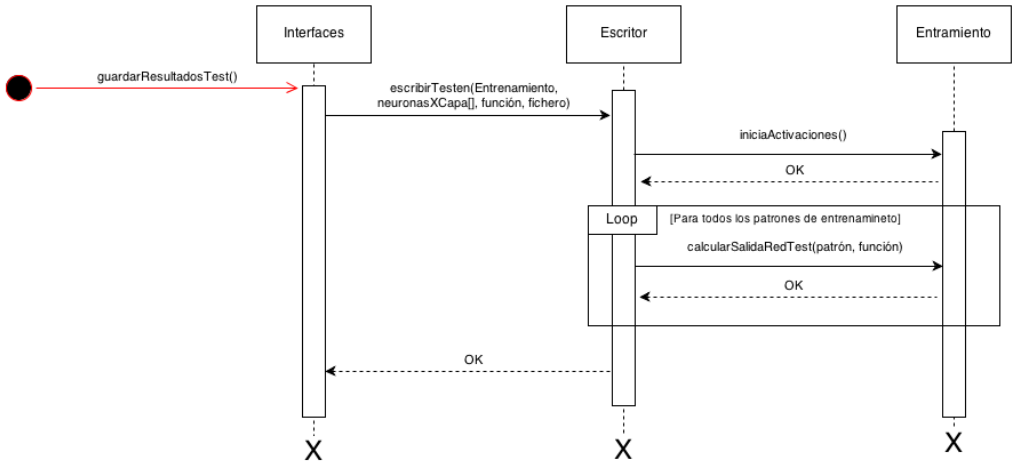

Ilustración 20: Iteraciones para el guardado de resultados para los patrones de test.

4.4. Diseño de clases

Una vez analizadas en profundidad las distintas clases y funcionalidades que componen el simulador, así como las iteraciones entre clases para cada una de las diferentes funcionalidades, es de rigor explicar el diseño en su conjunto. En la siguiente ilustración (ilustración 21) se muestra un diagrama de clases que detalla los atributos de las distintas clases, sus métodos u operaciones y como se relacionan entre sí.

El diseño del sistema, como se ha podido ver en las secciones anteriores, está compuesto por diferentes clases que interactúan entre sí para llevar a cabo las distintas funcionalidades del simulador. Estas clases tienen métodos muy específicos que llevan a cabo funciones muy concretas. El motivo de diseñar las clases de esta manera es, como se mencionaba anteriormente, enfocar el diseño de manera que facilite en la medida de lo posible su sencillez, comprensión y el desarrollo de futuras mejoras.

El sistema se compone de las siguientes clases:

- Proyecto: se encarga de iniciar el sistema.
- Interfaces: tiene como función la visualización la interfaz y la recogida de peticiones por parte del usuario, así como gestionar las llamadas a las funcionalidades que satisfagan las peticiones de éste.
- Arquitectura: su función es la definición y almacenamiento de la información referente a la arquitectura de red.
- Fichero: se encarga de todas aquellas operaciones relacionadas con la carga de ficheros de datos.
- Entrenamiento: es responsable de todas aquellas operaciones y funciones relacionadas con el proceso de aprendizaje de las arquitecturas de red, así como del cálculo de errores.
- Cargador: se encarga de todas aquellas funciones relacionadas con las carga de arquitecturas de red en el simulador.

- Escritor: es responsable de todas aquellas funciones relacionadas con el guardado de información en ficheros de texto, ya sean resultados obtenidos por la red o arquitecturas de red.
- Graficador: tiene como función definir y mostrar al usuario un gráfico con la evolución del error a lo largo del proceso de aprendizaje.

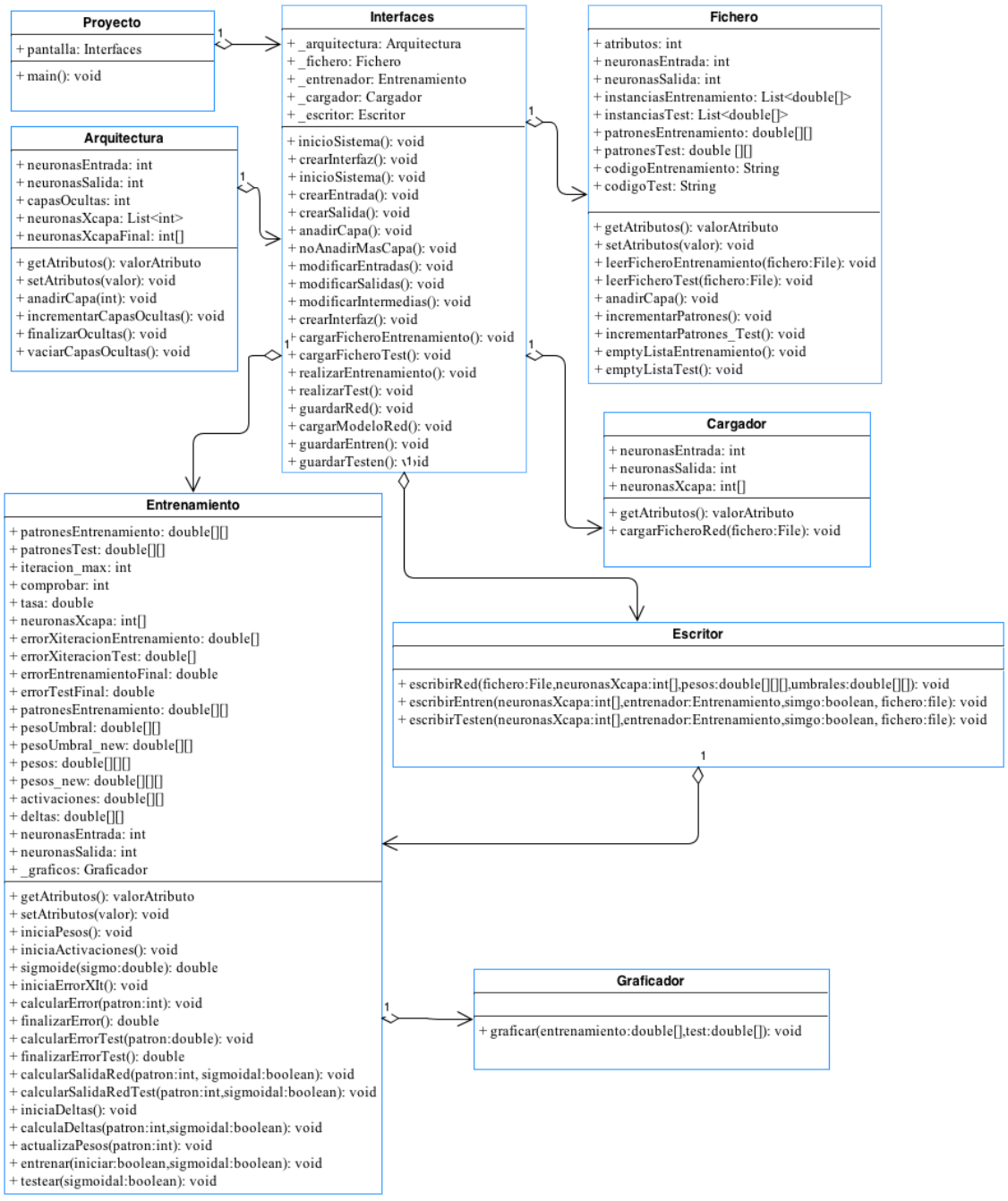

Ilustración 21: Diagrama de clases del sistema.

4.5. Especificaciones de excepciones

En el presente apartado se detallan las distintas excepciones o errores que puede capturar el simulador debido a un uso incorrecto del mismo por parte del usuario. Las excepciones que a continuación se detallan incluyen un identificador de excepción que tiene la siguiente nomenclatura, “EX-XX” donde “EX” es una abreviación de “Excepción” y “XX” un identificador numérico de dos dígitos.

EX-01	Valor no numérico en las neuronas
Descripción	El usuario introduce caracteres no numéricos al especificar las neuronas de una capa
Respuesta	Mensaje de error indicando: “El valor introducido para las neuronas ha de ser un número natural”

Tabla 66: EX-01 Valor no numérico en las neuronas.

EX-02	Valor nulo o negativo en las neuronas
Descripción	El usuario introduce un valor menor o igual que 0 al especificar las neuronas de una capa
Respuesta	Mensaje de error indicando: “El valor introducido para las neuronas ha de ser mayor de 0”

Tabla 67: EX-02 Valor nulo o negativo en las neuronas.

EX-03	Valor real en las neuronas
Descripción	El usuario introduce un número real en al especificar las neuronas de una capa
Respuesta	Mensaje de error indicando: “El valor introducido para las neuronas ha de ser mayor de 0”

Tabla 68: EX-03 Valor racional en las neuronas.

EX-04	Exceder el máximo de capas ocultas
Descripción	El usuario intenta agregar más de 10 capas ocultas.
Respuesta	Mensaje de error indicando: “Este simulador acepta un máximo de 10 capas ocultas”

Tabla 69: EX-04 Exceder el máximo de capas ocultas.

EX-05	Ausencia de cabecera en los ficheros de datos
Descripción	El usuario no añade la cabecera a los ficheros de datos de entrenamiento y test
Respuesta	Mensaje de error indicando: "Error en la lectura de la cabecera. Formato: atributos, entradas, salidas"

Tabla 70: EX-05 Ausencia de cabecera en los ficheros de datos.

EX-06	Error en la cabecera en los ficheros de datos
Descripción	El usuario añade la cabecera incorrectamente
Respuesta	Mensaje de error indicando: "Error en la cabecera: entradas más salidas distinto de atributos"

Tabla 71: EX-06 Error en la cabecera en los ficheros de datos.

EX-07	Error los patrones de datos
Descripción	Alguno de los patrones de datos del fichero que se esté cargando no coincide con la cabecera especificada, (faltan atributos, hay una línea en blanco).
Respuesta	Mensaje de error indicando: "Error en los patrones de datos"

Tabla 72: EX-07 Error los patrones de datos.

EX-08	Fichero de datos vacío
Descripción	El fichero de datos está vacío
Respuesta	Mensaje de error indicando: "El fichero se halla vacío"

Tabla 73: EX-08 Fichero de datos vacio.

EX-09	Valor erróneo en la tasa de aprendizaje
Descripción	El usuario introduce un valor no numérico, negativo o superior a 1 al especificar la tasa de aprendizaje
Respuesta	Mensaje de error indicando: "El valor introducido para la razón de aprendizaje ha de ser numérico y tener valor entre 0 y 1"

Tabla 74: EX-09 Valor erróneo en la tasa de aprendizaje.

EX-10	Valor erróneo en el número de iteraciones
Descripción	El usuario introduce un valor que no sea un número natural no nulo al especificar el número de iteraciones del proceso de aprendizaje
Respuesta	Mensaje de error indicando: "El valor introducido para el numero de iteraciones ha de ser un número natural mayor que 0"

Tabla 75: EX-10 Valor erróneo en la tasa de aprendizaje.

EX-11	Valor erróneo en las comprobaciones
Descripción	El usuario introduce un valor que no sea un número natural no nulo al especificar el intervalo de comprobaciones de evolución del error
Respuesta	Mensaje de error indicando: "El valor introducido para las comparaciones ha de ser un número natural mayor que 0"

Tabla 76: EX-11 Valor erróneo las comprobaciones.

EX-12	Iteraciones entre comprobaciones superior a iteraciones totales
Descripción	El usuario introduce un valor mayor para el intervalo entre comprobaciones de la evolución del error que el número total de iteraciones
Respuesta	Mensaje de error indicando: "El valor introducido para las comparaciones ha de ser menor que el numero de iteraciones"

Tabla 77: EX-12 Iteraciones entre comprobaciones superior a iteraciones totales.

EX-13	Diferentes neuronas de entrada
Descripción	Cuando el usuario entrene la red o calcule su error actual, el número de neuronas de entrada de la arquitectura no coincide con el de los ficheros de datos o las neuronas de entrada de los ficheros de datos no coinciden entre sí
Respuesta	Dependiendo de qué par no coincidan, se mostrarán los siguientes mensajes de error: • "El número de neuronas de entrada de la arquitectura es distinto de las del fichero de entrenamiento" • "El número de neuronas de entrada de la arquitectura es distinto de las del fichero de test" • "El número de neuronas de entrada de los ficheros de test y entrenamiento no coinciden"

Tabla 78: EX-13 Diferentes neuronas de entrada.

EX-14	Diferentes neuronas de salida
Descripción	Cuando el usuario entrene la red o calcule su error actual, el número de neuronas de salida de la arquitectura no coincide con el de los ficheros de datos o las neuronas de salida de los ficheros de datos no coinciden entre sí
Respuesta	Dependiendo de qué par no coincidan, se mostrarán los siguientes mensajes de error: • "El número de neuronas de salida de la arquitectura es distinto de las del fichero de entrenamiento" • "El número de neuronas de salida de la arquitectura es distinto de las del fichero de test" • "El número de neuronas de salida de los ficheros de test y entrenamiento no coinciden"

Tabla 79: EX-14 Diferentes neuronas de salida.

EX-15	No se especifican capas ocultas
Descripción	Cuando el usuario entrena la red y la arquitectura no cuenta con ninguna capa oculta
Respuesta	Mensaje de error indicando: "No hay ninguna capa oculta"

Tabla 80: EX-15 No se especifican capas ocultas.

EX-16	No se ha cerrado la configuración de las capas ocultas
Descripción	Cuando el usuario entrena la red y sigue habilitada la opción de añadir capas ocultas
Respuesta	Mensaje de error indicando: "Por favor antes de entrenar finalice la configuración de las capas ocultas"

Tabla 81: EX-16 No se cerrado la configuración de las capas ocultas.

EX-17	El fichero que se intenta cargar no existe
Descripción	Al cargar cualquier fichero de datos o de arquitectura de red se especifica un fichero inexistente
Respuesta	Mensaje de error indicando: "El fichero no existe"

Tabla 82: EX-17 El fichero que se intenta cargar no existe.

EX-18	Error en el guardado de datos
Descripción	Al guardar los resultados obtenidos sobre entrenamiento o test o una arquitectura de red se produce un error que impide que se guarden (por ejemplo por permisos de escritura en la carpeta de destino)
Respuesta	Dependiendo de qué fichero falle al guardarse, se mostrarán los siguientes mensajes de error: • "Se ha producido un error al guardar la red" • "Se ha producido un error al guardar los resultados de entrenamiento" • "Se ha producido un error al guardar los resultados de test"

Tabla 83: EX-18 Error en el guardado de datos.

EX-19	Error al cargar una arquitectura de red
Descripción	Al cargar una arquitectura de red hay algún error en el formato
Respuesta	Mensaje de error indicando: “Error en la lectura del fichero de carga”

Tabla 84: EX-19 Error al cargar una arquitectura.

4.6. Normativa de código

Para facilitar la comprensión del código, así como su modificación por aquellas personas que no han participado en su desarrollo, a continuación se detalla las distintas normativas, nomenclaturas y estándares que se han utilizado en el simulador.

En primer lugar, al comienzo de cada clase se especificará la persona responsable de su desarrollo, así como el nombre de la clase y una breve descripción de la misma, siguiendo los siguientes formatos:

- Autor: “//Autor: Nombre y Apellidos”
- Clase y descripción: “/**
  * Clase Nombre: descripción
  */”

Cada clase en el código fuente se representa en un fichero independiente, de tal modo que en un mismo fichero no coincidan las especificaciones de dos o más clases. Respecto al estándar para nombrar a las clases se ha utilizado el estilo “PascalCase” palabras o palabras compuestas con sus iniciales en mayúsculas, por ejemplo: Ejemplo o EjemploPractico.

Para el nombramiento de los métodos y variables se utiliza por el contrario el estándar “CamelCase” que utiliza palabras o palabras compuestas con las iniciales en mayúsculas a excepción de la primera palabra, por ejemplo: ejemplo o ejemploPractico.

En lo referente al estilo del código se han seguido las siguientes pautas:

- Los comentarios de línea vendrán precedidos de “//”.
- Los comentarios de líneas múltiples utilizarán “/**” en su apertura, “*” al comienzo de cada línea y “*/” en su clausura.
- Los comentarios de código aparecen situados a disposición del autor en las partes del código que considera necesarias debido a su extensión o complejidad.
- Sólo se declarara una variable por declaración.
- Se practicará la identación mediante tabulaciones compuestas por 4 espacios en blanco.
- Siempre se utilizará los caracteres de llave “{” y ”}” cuando sea opcional.
- Sólo habrá una llave por línea.
- La llave de apertura “{” se situará al final de la primera línea del bucle o condición y la llave de clausura “}” se situará en la siguiente a la última sentencia correspondiente al bucle o condición.

4.7. Entorno tecnológico

Para la correcta utilización y codificación del sistema es necesario que el entorno tecnológico que se esté utilizando satisfaga unos requisitos mínimos necesarios. A continuación se detallan dichos requisitos tanto a nivel de Hardware como de SoftWare.

4.7.1. Hardware

A nivel de HardWare los requisitos que ha de satisfacer el entorno tecnológico son comunes tanto para su utilización como para su codificación. Los requisitos mínimos que ha de satisfacer son los siguientes:

- Contar con un procesador Pentium IV. De 2.5 GHz o superior.
- Tener al menos 1 GB de memoria RAM.
- Tener al menos 2 GB de espacio en disco duro.
- Contar con un monitor, ratón y teclado.

4.7.2. Software

A nivel de SoftWare sí se diferencia entre los requisitos mínimos necesarios para su utilización y codificación. Los requisitos mínimos son los siguientes:

- Para su codificación será necesario que el entorno satisfaga los siguientes requisitos:
    - El sistema operativo será Microsoft Windows Vista o superior.
    - Tener instalada la máquina virtual de Java.
    - Tener instalado Java SE 6 o superior.
    - Utilizar como entorno de desarrollo Eclipse Indigo (también conocido como Eclipse 3.7) o superior.
    - Para la edición de la documentación se utilizara Microsoft Office 2007 o superior.

- Para su utilización el único requisito necesario será tener instalada la máquina virtual de Java, esto se debe a las características de portabilidad de Java y a su independencia del sistema operativo.

Capítulo 5. Pruebas Experimentales

A lo largo del presente capítulo se procederá a especificar las pruebas que se han llevado a cabo sobre el simulador desarrollado para su validación y su trazabilidad o relación con los requisitos de Software anteriormente detallados. Para la realización de estas pruebas se han utilizado diferentes dominios que se explicarán a lo largo del capítulo. Por último, se procederá a abordar un problema real utilizando el simulador.

5.1. Definición de las pruebas

A continuación se procede a detallar las distintas pruebas que se han realizado para validar el correcto funcionamiento del simulador. Estas pruebas se han determinado en función de las funcionalidades y objetivos que se fijaron al comienzo del proyecto y su objetivo es comprobar que se ha cumplido satisfactoriamente cada una de ellos.

Para cada una de las pruebas que se especifican a continuación se detalla la siguiente información:

- Identificador unívoco con la siguiente nomenclatura “PR-XX” donde PR es una abreviatura de “Prueba” y “XX” un identificador numérico de dos dígitos.
- Descripción de la prueba.
- Estado inicial.
- Estado final deseado.
- Dependencia respecto de los requisitos.

Identificador	PR-01
Descripción	Definición de una arquitectura de red.
Estado inicial	Simulador iniciado.
Estado final	Se ha definido la siguiente arquitectura: • Capa de entrada: 1 neurona. • Capa oculta 1: 5 neuronas. • Capa oculta 2: 3 neuronas • Capa de salida: 1 neurona.
Dependencias de requisitos	RSF-01

Tabla 85: Prueba experimental PR-01.

Identificador	PR-02
Descripción	Modificación de una arquitectura de red.
Estado inicial	Arquitectura de red definida con la siguiente arquitectura: • Capa de entrada: 1 neurona. • Capa oculta 1: 5 neuronas. • Capa oculta 2: 3 neuronas • Capa de salida: 1 neurona.
Estado final	Se ha modificado la arquitectura de red a la siguiente configuración: • Capa de entrada: 4 neuronas. • Capa oculta 1: 3 neuronas. • Capa de salida: 3 neurona.
Dependencias de requisitos	RSF-01, RSF-02

Tabla 86: Prueba experimental PR-02.

Identificador	PR-03
Descripción	Carga de ficheros de datos de entrenamiento.
Estado inicial	Se ha definido una arquitectura de red.
Estado final	Si el fichero cargado tiene el formato correcto el simulador lo interpretará y cargará correctamente, en caso contrario devolverá el mensaje de error correspondiente al fichero.
Dependencias de requisitos	RSF-01, RSF-03

Tabla 87: Prueba experimental PR-03.

Identificador	PR-04
Descripción	Carga de ficheros de datos de test.
Estado inicial	Se ha definido una arquitectura de red y cargado un fichero de entrenamiento.
Estado final	Si el fichero cargado tiene el formato correcto el simulador lo interpretará y cargará correctamente, en caso contrario devolverá el mensaje de error correspondiente al fichero.
Dependencias de requisitos	RSF-01, RSF-04

Tabla 88: Prueba experimental PR-04.

Identificador	PR-05
Descripción	Selección de la función de activación.
Estado inicial	Se define una arquitectura de red, se cargan los ficheros de datos, se definen los parámetros de ejecución del entrenamiento, se entrena la red.
Estado final	Se guardan las salidas obtenidas y se comprueba que se haya aplicado la función de activación de la salida.
Dependencias de requisitos	RSF-01, RSF-05, RSF-10

Tabla 89: Prueba experimental PR-05.

Identificador	PR-06
Descripción	Definición de la tasa de aprendizaje.
Estado inicial	Se define una arquitectura de red, se cargan los ficheros de datos, se definen los parámetros de entrenamiento.
Estado final	Se ejecuta el entrenamiento con varias tasas de aprendizajes y valores incorrectos. Se comparan los errores obtenidos para las distintas tasas de aprendizaje y que el simulador devuelva el error correcto para los valores incorrectos.
Dependencias de requisitos	RSF-01, RSF-06, RSF-10

Tabla 90: Prueba experimental PR-06.

Identificador	PR-07
Descripción	Definición del número de iteraciones.
Estado inicial	Se define una arquitectura de red, se cargan los ficheros de datos, se definen los parámetros de entrenamiento.
Estado final	Se ejecuta el entrenamiento con diferentes iteraciones máximas y valores incorrectos. Se comparan los errores obtenidos para los distintos números de iteraciones y que el simulador devuelva el error correcto para los valores incorrectos.
Dependencias de requisitos	RSF-01, RSF-07, RSF-10

Tabla 91: Prueba experimental PR-07.

Identificador	PR-08
Descripción	Definición del número de iteraciones entre comprobaciones de evolución del error.
Estado inicial	Se define una arquitectura de red, se cargan los ficheros de datos, se definen los parámetros de entrenamiento.
Estado final	Se ejecuta el entrenamiento con diferentes números de iteraciones entre comprobaciones de evolución del error y valores incorrectos. Se comprueba en la gráfica de evolución del error observando el eje X cuántas veces se ha comprobado el error durante el entrenamiento.
Dependencias de requisitos	RSF-01, RSF-08, RSF-10

Tabla 92: Prueba experimental PR-08.

Identificador	PR-09
Descripción	Selección del método de inicialización de los pesos.
Estado inicial	Se carga una red ya entrenada.
Estado final	Se selecciona el método de inicialización de “Utilizar pesos actuales” y se entrena la red, se comprueba que el error inicial del nuevo entrenamiento es el error final del entrenamiento anterior.
Dependencias de requisitos	RSF-01, RSF-09, RSF-10

Tabla 93: Prueba experimental PR-09.

Identificador	PR-10
Descripción	Entrenamiento de la red.
Estado inicial	Se define una arquitectura de red, se cargan los ficheros de datos, se definen los parámetros de entrenamiento y se entrena la red.
Estado final	Se comprueban que las salidas obtenidas aproximen correctamente las deseadas.
Dependencias de requisitos	RSF-01, RSF-03, RSF-04, RSF-05, RSF-06, RSF-07, RSF-08, RSF-09, RSF-10

Tabla 94: Prueba experimental PR-10.

Identificador	PR-11
Descripción	Comprobación del error actual de la red.
Estado inicial	Se define una arquitectura de red, se cargan los ficheros de datos, se definen los parámetros de entrenamiento y se entrena la red.
Estado final	Se calcula el error actual de la red y se comprueba que se corresponda con el error devuelto por el entrenamiento de la red.
Dependencias de requisitos	RSF-01, RSF-03, RSF-04, RSF-05, RSF-06, RSF-07, RSF-08, RSF-09, RSF-10, RSF-11

Tabla 95: Prueba experimental PR-11.

Identificador	PR-12
Descripción	Guardar la red actual.
Estado inicial	Se define una arquitectura de red, se cargan los ficheros de datos, se definen los parámetros de entrenamiento y se entrena la red.
Estado final	Se guarda la red actual en un fichero y se comprueba que tenga el formato correcto.
Dependencias de requisitos	RSF-01, RSF-03, RSF-04, RSF-05, RSF-06, RSF-07, RSF-08, RSF-09, RSF-10, RSF-13

Tabla 96: Prueba experimental PR-12.

Identificador	PR-13
Descripción	Cargar una arquitectura de red.
Estado inicial	Se inicia el simulador.
Estado final	Se carga la red y se comprueba que la arquitectura se define correctamente en el simulador.
Dependencias de requisitos	RSF-01, RSF-14

Tabla 97: Prueba experimental PR-13.

Identificador	PR-14
Descripción	Guardar los resultados de red.
Estado inicial	Se define una arquitectura de red, se cargan los ficheros de datos, se definen los parámetros de entrenamiento y se entrena la red.
Estado final	Se guardan los resultados de la red en un fichero de texto y se comprueba que estén correctamente aproximados y que el fichero tenga el formato correcto.
Dependencias de requisitos	RSF-01, RSF-03, RSF-04, RSF-05, RSF-06, RSF-07, RSF-08, RSF-09, RSF-10, RSF-12

Tabla 98: Prueba experimental PR-14.

5.2. Trazabilidad de pruebas con los requisitos

A continuación se detalla la correlación o dependencia entre las pruebas anteriormente detalladas y los requisitos de Software establecidos en el sub-apartado 3.3.2.

	RSF-01	RSF-02	RSF-03	RSF-04	RSF-05	RSF-06	RSF-07	RSF-08	RSF-09	RSF-10	RSF-11	RSF-12	RSF-13	RSF-14
PR-01	X													
PR-02	X	X												
PR-03	X		X											
PR-04	X			X										
PR-05	X				X					X				
PR-06	X					X				X				
PR-07	X						X			X				
PR-08	X							X		X				
PR-09	X								X	X				
PR-10	X		X	X	X	X	X	X	X	X				
PR-11	X		X	X	X	X	X	X	X	X	X			
PR-12	X		X	X	X	X	X	X	X	X			X	
PR-13	X													X
PR-14	X		X	X	X	X	X	X	X	X		X		

Tabla 99: Matriz de trazabilidad entre pruebas y requisitos.

5.3. Dominios utilizados: definición y resultados

En el presente apartado se definirán los dominios que se han utilizado para la realización de las pruebas para la validación del sistema. Se destaca que el simulador pasó de manera satisfactoria las distintas pruebas anteriormente especificadas para cada uno de los dominios que a continuación se definen.

Para comprobar el correcto funcionamiento del simulador se han elegido 3 dominios con características diferentes. El primero de ellos, el polinomio de Hermite, es un problema de aproximación o regresión que se genera artificialmente y está formado por 1 variable de entrada y 1 variable de salida. El segundo, el dominio de Housing, se ha elegido por ser un problema de aproximación con mayor número de variables de entrada, en este caso, 13. Y finalmente, se ha elegido un dominio de clasificación, Balance-Scale, que posee 3 variables de salida, a diferencia de los dominios anteriores que tienen únicamente una variable de salida

Junto a la definición de estos dominios, se incluirán los resultados obtenidos en la prueba 10 (Tabla 94). Los resultados de esta prueba validan el correcto funcionamiento del simulador comprobando que se haya realizado de manera correcta el aprendizaje de la red.

Para cada uno de los dominios se ha realizado numerosas veces esta prueba utilizando diferentes arquitecturas de red y tasas de aprendizaje, pero se han escogido 15 de entre éstas, para compararlas entre sí, diferenciando 5 arquitecturas de red y tres tasas de aprendizaje diferentes.

5.3.1. Polinomio de Hermite

El primer dominio utilizado para validar el sistema es el polinomio de Hermite. El polinomio de Hermite es un ejemplo de polinomio ortogonal utilizado en Probabilística. Los patrones de datos de este dominio están compuestos por 1 entrada y 1 salida.

Para generar los datos se ha utilizado la siguiente función:

$$f(x) = 1.1\,(1 - x + 2x^2)e^{-(1/2)x^2}$$

Ecuación 16: Polinomio de Hermite

Como valores de entrada generar los datos se ha utilizado una distribución uniforme de valores aleatorios pertenecientes al intervalo [-4,4]. Una vez generado el conjunto de datos se ha normalizado al intervalo [0,1] y se ha dividido en dos subconjuntos para su uso como conjuntos de entrenamiento y test compuestos por 40 y 200 patrones, respectivamente.

Las arquitecturas de red que se han utilizado paras las pruebas en el dominio de Hermite son las siguientes:

- Arquitectura 1:
  - Capa de entrada: 1 neurona.
  - Capa oculta 1: 1 neurona.
  - Capa de salida: 1 neurona.
- Arquitectura 2:
  - Capa de entrada: 1 neurona.
  - Capa oculta 1: 5 neuronas.
  - Capa de salida: 1 neurona.
- Arquitectura 3:
  - Capa de entrada: 1 neurona.
  - Capa oculta 1: 10 neuronas.
  - Capa de salida: 1 neurona.
- Arquitectura 4:
  - Capa de entrada: 1 neurona.
  - Capa oculta 1: 3 neuronas.

  - Capa oculta 2: 4 neuronas.
  - Capa de salida: 1 neurona.
- Arquitectura 5:
  - Capa de entrada: 1 neurona.
  - Capa oculta 1: 5 neuronas.
  - Capa oculta 2: 3 neuronas.
  - Capa de salida: 1 neurona.

A continuación se muestran los resultados obtenidos en cada una de las pruebas realizadas con las arquitecturas anteriores.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 1	0.01	10000	0.08612600	0.08432838
Arquitectura 1	0.1	10000	0.06900165	0.06615330
Arquitectura 1	0.2	10000	0.06906439	0.06599424
Arquitectura 2	0.01	10000	0.09008157	0.08888346
Arquitectura 2	0.1	10000	0.01503935	0.01520802
Arquitectura 2	0.2	10000	0.00886358	0.00888088
Arquitectura 3	0.01	10000	0.07888514	0.07753165
Arquitectura 3	0.1	10000	0.01100012	0.01107599
Arquitectura 3	0.2	10000	0.01389814	0.01403876
Arquitectura 4	0.01	10000	0.09109138	0.09069071
Arquitectura 4	0.1	10000	0.00474316	0.00471214
Arquitectura 4	0.2	10000	$9.5820E^{-4}$	$9.1855E^{-4}$
Arquitectura 5	0.01	10000	0.09077682	0.09027663
Arquitectura 5	0.1	10000	0.01176226	0.01192718
Arquitectura 5	0.2	10000	$5.7826E^{-4}$	$5.6915E^{-4}$

Tabla 100: Resultados del dominio de Hermite.

Una vez ejecutadas las pruebas se pueden comparar los errores obtenidos para los conjuntos de datos de entrenamiento y test, de esta manera se puede decidir qué arquitectura de red es más adecuada para el dominio del polinomio de Hermite.

Para comprobar el correcto funcionamiento del aprendizaje y observar la capacidad de aproximación de la red, a continuación se muestran dos gráficas (ilustración 22 e ilustración 23) que muestran una comparativa entre las salidas deseadas para los patrones de entrenamiento y test y las que se han obtenido con la red.

Las siguientes gráficas presentan el siguiente formato:

- El eje Y representa el valor de la salida.
- El eje X el número del patrón de datos al que corresponde.

A continuación se puede observar la comparativa entre las salidas deseadas y obtenidas para los patrones de entrenamiento:

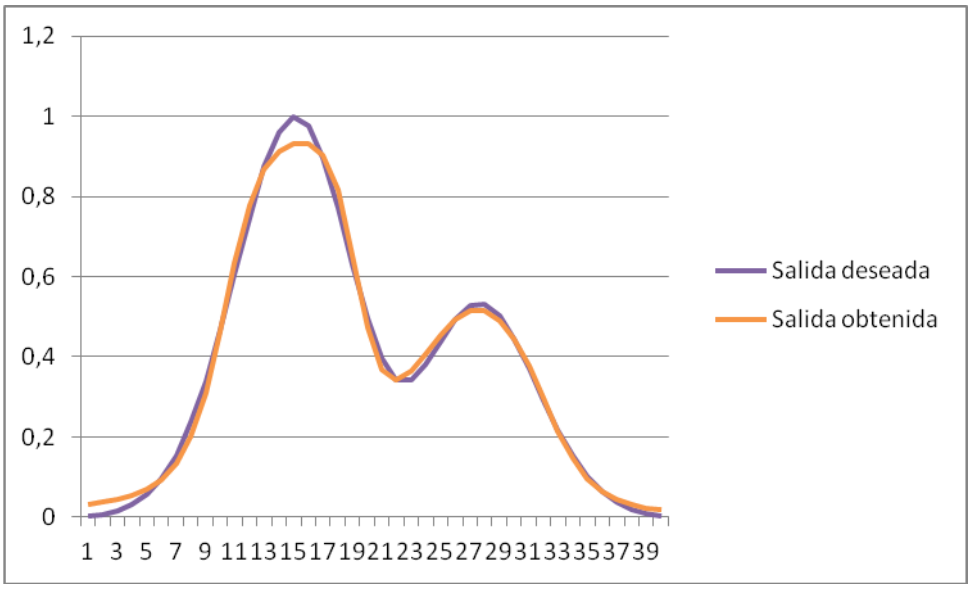

Ilustración 22: Comparativa entre salidas deseadas y obtenidas de Hermite (entrenamiento).

A continuación se puede observar la comparativa entre las salidas deseadas y obtenidas para los patrones de test:

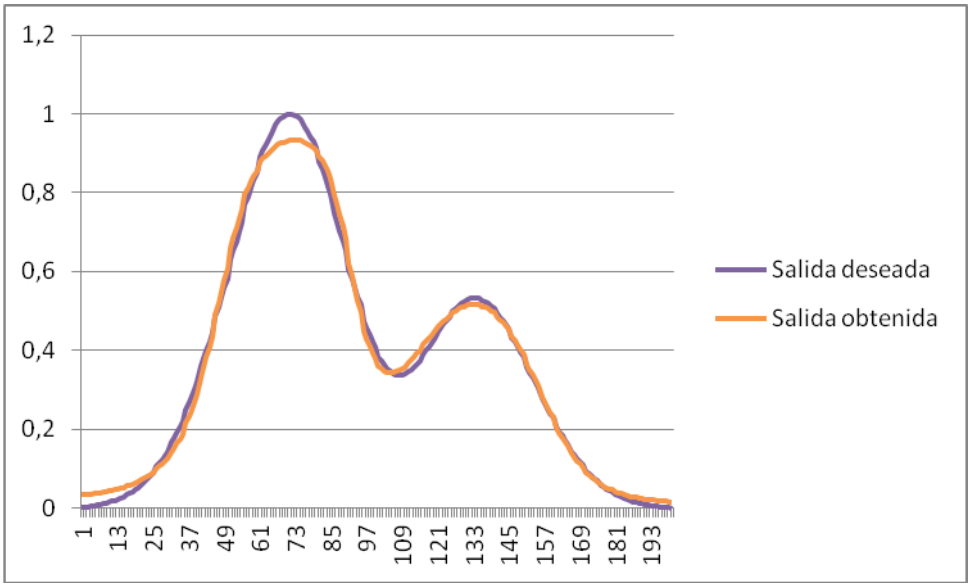

Ilustración 23: Comparativa entre salidas deseadas y obtenidas de Hermite (test).

En las gráficas anteriores se puede apreciar como las salidas obtenidas con la red se aproximan a la salida deseada, cometiéndose un mayor error para las salidas cercanas a 0 ó a 1. Esto se debe a que se ha elegido como función de activación de salida de la red la función sigmoidal, la cual hace que en ocasiones la red tenga dificultad para aproximar los valores extremos. Este comportamiento es habitual en el Perceptron Multicapa. Ante esta situación se da por

validado el funcionamiento del simulador para el domino de prueba del polinomio de Hermite, caracterizado por una entrada y una salida.

5.3.2. Dominio de Housing

El dominio de Housing es un problema basado en datos reales para la predicción del precio de las casas en los suburbios de Boston. Los datos que componen este dominio se han obtenido de UCI Machine Learning Repository [2].

Los patrones del dominio de Housing están compuestos por 14 atributos de los cuales los 13 primeros se corresponden con las variables de entrada del problema y el último es la salida del problema, es decir el precio estimado para la casa. Para la validación del sistema se han formado dos conjuntos de datos, el primero de 70 patrones datos para entrenamiento y el segundo de 81 patrones para test. Una vez definidos los conjuntos de datos, estos se han modelado estos datos y normalizado al intervalo [0,1].

Las arquitecturas de red que se han utilizado paras las pruebas en el dominio de Housing son las siguientes:

- Arquitectura 1:
  - Capa de entrada: 13 neuronas.
  - Capa oculta 1: 1 neurona.
  - Capa de salida: 1 neurona.
- Arquitectura 2:
  - Capa de entrada: 13 neuronas.
  - Capa oculta 1: 5 neuronas.
  - Capa de salida: 1 neurona.
- Arquitectura 3:
  - Capa de entrada: 13 neuronas.
  - Capa oculta 1: 10 neuronas.
  - Capa de salida: 1 neurona.
- Arquitectura 4:
  - Capa de entrada: 13 neuronas.
  - Capa oculta 1: 3 neuronas.
  - Capa oculta 2: 4 neuronas.
  - Capa de salida: 1 neurona.
- Arquitectura 5:
  - Capa de entrada: 13 neuronas.
  - Capa oculta 1: 5 neuronas.
  - Capa oculta 2: 3 neuronas.
  - Capa de salida: 1 neurona.

A continuación se muestran los resultados obtenidos en cada una de las pruebas realizadas con las arquitecturas anteriores.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 1	0.01	10000	0.00822902	0.01363200
Arquitectura 1	0.1	10000	0.00789909	0.01415008
Arquitectura 1	0.2	10000	0.00804137	0.01396473
Arquitectura 2	0.01	10000	0.00645504	0.01420471
Arquitectura 2	0.1	10000	0.00135046	0.01515371
Arquitectura 2	0.2	10000	0.00102949	0.01377694
Arquitectura 3	0.01	10000	0.00529735	0.01239528
Arquitectura 3	0.1	10000	0.00123726	0.01531982
Arquitectura 3	0.2	10000	$9.9572E^{-4}$	0.01493344
Arquitectura 4	0.01	10000	0.00761521	0.01347810
Arquitectura 4	0.1	10000	0.00190023	0.01612292
Arquitectura 4	0.2	10000	0.00139315	0.01927311
Arquitectura 5	0.01	10000	0.00753443	0.01371102
Arquitectura 5	0.1	10000	0.00168223	0.01385794
Arquitectura 5	0.2	10000	0.00121326	0.01613077

Tabla 101: Resultados del dominio de Housing.

Los resultados obtenidos en esta prueba resultan interesantes por las conclusiones que se puede extraer de la misma. Respecto a la razón de aprendizaje se puede observar que si la que si esta es pequeña el error de entrenamiento es mayor, mientras que si la razón de aprendizaje aumenta el error de entrenamiento se reduce debido a una convergencia más rápida, provocando que el error de test sea mayor.

Otra conclusión interesante que se pudo observar durante el proceso de entrenamiento, es que en este problema, al contrario de lo que sucedió con el dominio de Hermite, se produjo sobre-aprendizaje. En la siguiente ilustración se puede observar la evolución del error a lo largo del proceso de aprendizaje, en la cual se puede observar el sobre-aprendizaje:

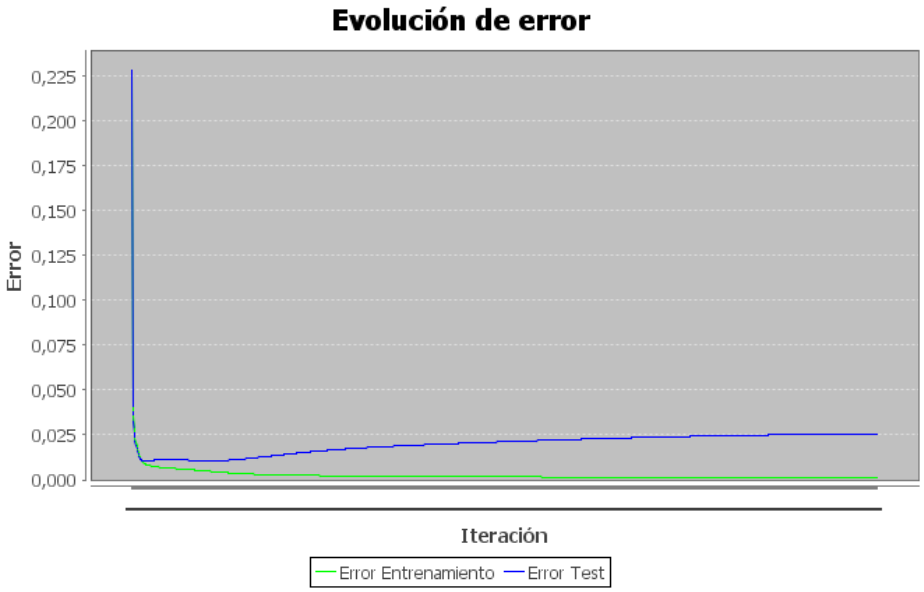

Ilustración 24: Sobre-aprendizaje en el dominio de Housing.

Para solucionar el sobre-aprendizaje se reentrenó la red reduciendo el número de iteraciones.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 2	0.2	8000	0.00102749	0.01277694
Arquitectura 3	0.2	8000	0.00102175	0.01590636

Tabla 102: Mejores resultado de Housing recalculado.

Para comprobar el correcto funcionamiento del aprendizaje y observar la capacidad de aproximación de la red, se muestran dos gráficas (ilustración 25 e ilustración 26) que muestran una comparativa entre las salidas deseadas para los patrones de entrenamiento y test y las que se han obtenido con la red.

Las siguientes gráficas presentan el siguiente formato:

Las siguientes gráficas presentan el siguiente formato:

- El eje Y representa el valor de la salida.
- El eje X el número del patrón de datos al que corresponde.

A continuación se puede observar la comparativa entre las salidas deseadas y obtenidas para los patrones de entrenamiento:

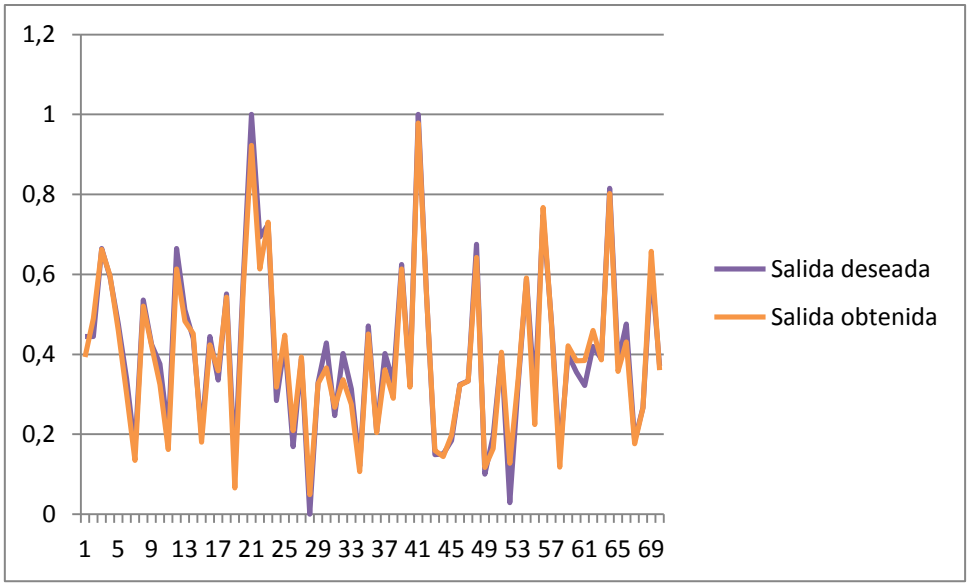

Ilustración 25: Comparativa entre salidas deseadas y obtenidas de Housing (entrenamiento).

A continuación se puede observar la comparativa entre las salidas deseadas y obtenidas para los patrones de test:

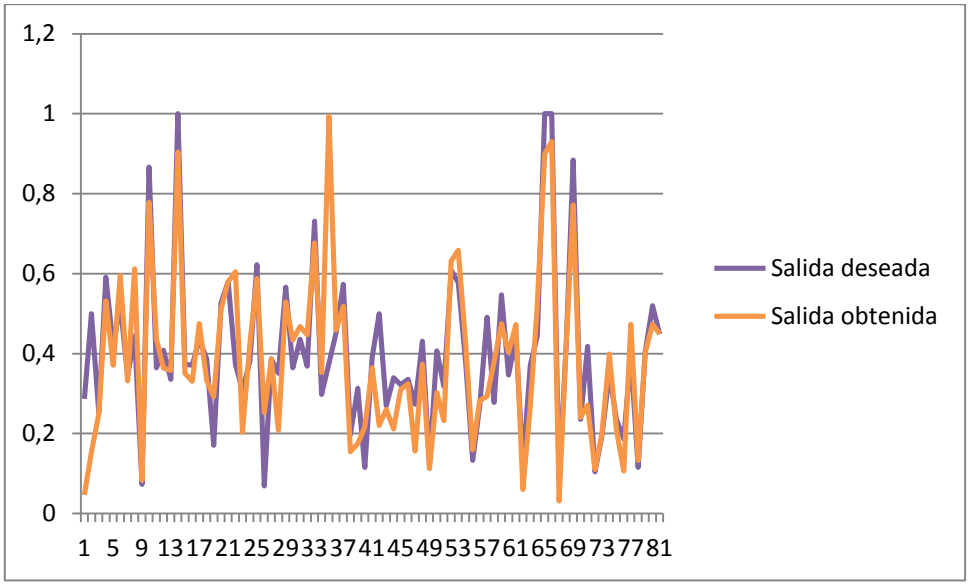

Ilustración 26: Comparativa entre salidas deseadas y obtenidas de Hermite (test).

En las gráficas anteriores se puede apreciar que los resultados se corresponden con los esperados por parte del Perceptrón Multicapa.

5.3.3. Dominio de la Balance Scale

El dominio de Balance Scale es un problema basado en datos reales para la clasificación del estado de una balanza en función de 4 pesas. Los datos que componen este dominio se han obtenido de UCI Machine Learning Repository [2].

Los patrones del dominio de Balance Scale están compuestos por 5 atributos de los cuales los 4 primeros se corresponden con las variables de entrada del problema, es decir con las pesas y el último es la salida del problema, es decir el estado de la balanza. La salida tiene tres posibles estados diferentes, por lo cual se trata de un problema de clasificación de 3 clases, para resolver el problema se ha transformado la salida utilizando 3 variables de salidas que representan cada clase. (Clase 1: (1 0 0), Clase 2: (0 1 0) y clase 3: (0 0 1)).

El conjunto de datos se compone de 625 entradas. Para la validación del sistema se han seleccionado 500 datos para el conjunto de entrenamiento y 125 para el conjunto de test. Una vez definidos los conjuntos de datos, estos se han normalizado en el al intervalo [0,1].

Las arquitecturas de red que se han utilizado paras las pruebas en el dominio de Balance Scale son las siguientes:

- Arquitectura 1:
  - Capa de entrada: 4 neuronas.
  - Capa oculta 1: 1 neurona.
  - Capa de salida: 3 neuronas.
- Arquitectura 2:
  - Capa de entrada: 4 neuronas.
  - Capa oculta 1: 5 neuronas.
  - Capa de salida: 3 neuronas.
- Arquitectura 3:
  - Capa de entrada: 4 neuronas.
  - Capa oculta 1: 10 neuronas.
  - Capa de salida: 3 neuronas.
- Arquitectura 4:
  - Capa de entrada: 4 neuronas.
  - Capa oculta 1: 3 neuronas.
  - Capa oculta 2: 4 neuronas.
  - Capa de salida: 3 neuronas.
- Arquitectura 5:
  - Capa de entrada: 4 neuronas.
  - Capa oculta 1: 5 neuronas.
  - Capa oculta 2: 3 neuronas.
  - Capa de salida: 3 neuronas.

A continuación se muestran los resultados obtenidos en cada una de las pruebas realizadas con las arquitecturas anteriores.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 1	0.01	10000	0.16348852	0.10464312
Arquitectura 1	0.1	10000	0.16514385	0.10564824
Arquitectura 1	0.2	10000	0.16740238	0.10711074
Arquitectura 2	0.01	10000	0.07864734	0.04040910
Arquitectura 2	0.1	10000	0.05423761	0.03458040
Arquitectura 2	0.2	10000	0.05754185	0.03613572
Arquitectura 3	0.01	10000	0.05594371	0.04506768
Arquitectura 3	0.1	10000	0.03199698	0.06005928
Arquitectura 3	0.2	10000	0.01666201	0.07038160
Arquitectura 4	0.01	10000	0.03711437	0.01877961
Arquitectura 4	0.1	10000	0.03899114	0.07012490
Arquitectura 4	0.2	10000	0.04457747	0.02268004
Arquitectura 5	0.01	10000	0.00636788	0.01775457
Arquitectura 5	0.1	10000	0.00405801	0.01947910
Arquitectura 5	0.2	10000	0.00406097	0.03752757

Tabla 103: Resultados del dominio de Balance Scale.

En los resultados anteriores se puede observar que la Arquitectura 1 que posee una única neurona oculta no es suficiente para clasificar correctamente los patrones de datos, mientras que según aumentamos el número de neuronas y de capas los resultados mejoran reduciéndose el error obtenido. También se puede observar que al contrario de lo que sucedía en el dominio de Housing, en el dominio de Balance Scale se obtienen mejores resultados con razones de aprendizaje menores, aunque su convergencia es más lenta.

Para comprobar la capacidad de la red para resolver el problema de clasificación, a continuación se muestran dos gráficas (ilustración 27 e ilustración 28) que muestran los porcentajes de acierto obtenidos, sobre los conjuntos de entrenamiento y test para la mejor de las arquitecturas anteriores. Se muestran los % totales, así como los % de clasificación en cada una de las clases.

A continuación se pueden observar la las tasas de acierto para los patrones de entrenamiento:

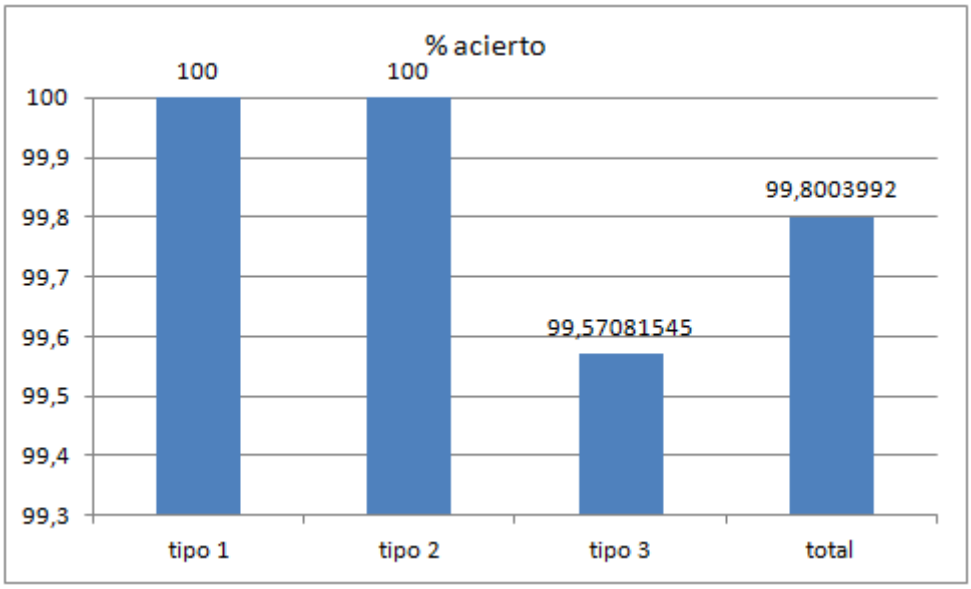

Ilustración 27: Tasas de acierto para el dominio de Balance (entrenamiento).

A continuación se pueden observar la las tasas de acierto para los patrones de test:

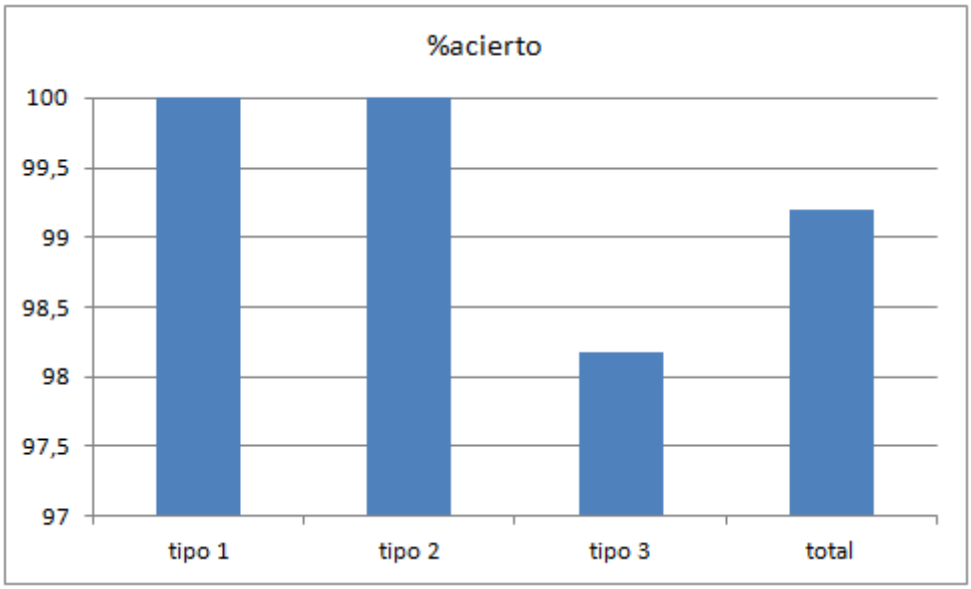

Ilustración 28: Tasas de acierto para el dominio de Balance (test).

Se puede observar que la red clasifica correctamente los patrones de la clase 1 y 2, pero tiene un porcentaje de acierto menor para la clase 3, este se debe a que el dominio de Balance Sacale es un dominio desbalanceado, es decir que posee menos ejemplos de una de las clases

5.4. Aplicación del Perceptrón Multicapa para la predicción de producción de huevos en granjas avícolas

En la presente sección se procederá a la resolución de un problema utilizando el simulador. Esta sección no tiene como objetivo único la validación del sistema, sino que tiene como objetivo abordar un problema de predicción utilizando el Percpetron Multicapa, para lo que se hará uso del simulador. Más allá de la validación del sistema, el propósito es estudiar y analizar la capacidad del Perceptron Multicapa para predecir la producción de huevos diaria.

5.4.1. Descripción del problema

El problema que se tratará de resolver a lo largo de esta sección es, como indica el título de la misma, la predicción de la producción de huevos en granjas avícolas utilizando la RNA del Perceptrón Multicapa para ello.

Este problema más específicamente trata de predecir la producción de huevos en una granja de Colombia, la Hacienda La Montaña, propiedad de la Universidad de Antioquia, localizada en el municipio de San Pedro de Los Milagros (Antioquia). El problema utiliza como datos la edad de un lote de gallinas y la producción de huevos de dicho lote día a día durante su estancia en la granja. El lote de gallinas utilizado para generar los datos del problema se trata de un lote de la raza H&N Brown Nick y está compuesto aproximadamente por 6400 gallinas.

El problema se va a abordar como un problema de predicción de series temporales, y el objetivo en este trabajo será la predicción de la producción de huevos en $t + 1$ y $t + 2$. Esto quiere decir, que dado un día t y la producción de huevos durante una serie de días pasados, se intentará predecir de manera independiente los huevos que se producirán los dos días siguientes al presente t ($t + 1$ y $t + 2$).

En el contexto de la producción de huevos uno de los modelos generalmente utilizados en el es el llamado modelo Lokhorst [3] que básicamente realiza una aproximación polinomial. El uso de dicho modelo requiere del ajuste manual de parámetros lo cual puede llegar a ser un trabajo laborioso. Este modelo además no es capaz de predecir las oscilaciones en la producción de huevos, quedándose próxima a una línea de tendencia.

5.4.2. Pre-procesado de datos

Como se menciona anteriormente los datos con lo que cuenta el problema son la edad de las gallinas de un lote en días y la producción de huevos del lote de cada día durante su estancia en la granja. Se trabajará con un conjunto de 281 datos, que corresponden con la producción de 281 días (desde el día 145 hasta el día 426). En la ilustración 29 se muestra la evolución de la producción respecto a la edad.

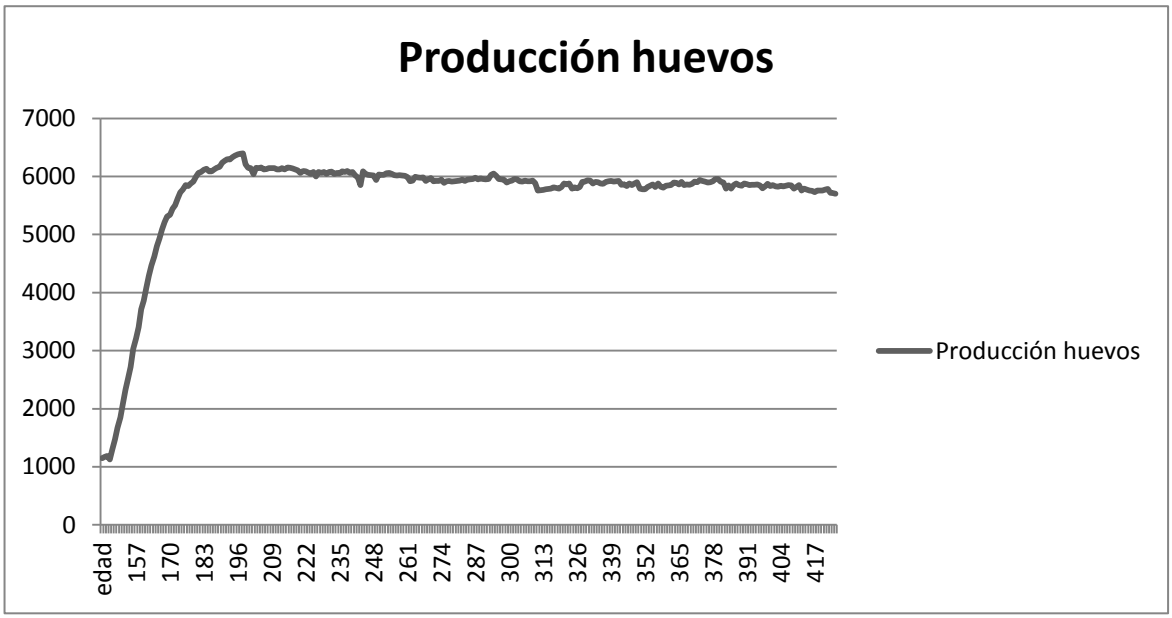

Ilustración 29: Producción de huevos frente a edad.

El fichero de datos presenta el siguiente formato:

edad	hvos
145	1150
146	1170
147	1190
148	1127
149	1305
150	1470
151	1673
152	1850

Ilustración 30: Datos originales del problema de predicción de huevos.

Estos datos deben ser transformados para abordar el problema desde un punto de vista de predicción como series temporales, siendo necesario generar nuevos patrones de datos que se adapten a los requerimientos del objetivo que se plantea alcanzar.

Este procesado de datos tiene como objetivo generar los conjuntos de datos que se van a utilizar para entrenamiento y test para cada de las pruebas que se van a realizar durante la resolución del problema. Para que los patrones de datos se adapten a los objetivos del problema, estos deberán tener los siguientes atributos:

- Edad de las gallinas.
- Producción de huevos en los n días anteriores, donde n será el número de días que se pretende utilizar como entrada del problema.
- Producción de huevos el día que se pretende predecir.

Una vez determinado el formato que han de tener los patrones, hay que decidir el valor de n (número de días tomado como entrada) que se va a utilizar. Para el presente problema se ha decidido utilizar la producción en los 3, 4 y 5 días anteriores. Y como se indicaba en la sección anterior se pretende predecir la producción en los días ($t+1$ y $t+2$).

Una vez generados los patrones de datos, se han normalizado todos los atributos al intervalo [0,1] utilizando la ecuación 17 y se ha aleatorizado el orden de los mismos. Por último, se ha dividido los conjuntos de datos en dos subconjuntos para entrenamiento y test, utilizando el 30% de los datos disponible para realizar el test.

Una vez finalizado el pre-procesado de datos, se tendrán seis diferentes conjuntos de entrenamiento y seis de test, los cuales presentan los siguientes formatos:

- Patrón con tres días anteriores, objetivo $t+1$:

edad	Prod(t-3)	Prod(t-2)	Prod(t-1)	Prod(t+1)
0.918149466	0.895236288	0.8861264	0.892579237	0.9003606
0.62633452	0.88308977	0.884608085	0.884987664	0.889163029
0.572953737	0.904915544	0.90871133	0.911178592	0.916302904

Ilustración 31: Patrones problema n=3 y objetivo t+1.

- Patrón con cuatro días anteriores, objetivo $t+1$:

edad	Prod(t-4)	Prod(t-3)	Prod(t-2)	Prod(t-1)	Prod(t+1)
0.106761566	0.775669007	0.794078573	0.799582463	0.819510344	0.830897704
0.387900356	0.929018789	0.927690264	0.913456064	0.931296261	0.929777946
0.288256228	0.944771304	0.936610362	0.942304043	0.941924464	0.937559309

Ilustración 32: Patrones problema n=4 y objetivo t+1.

- Patrón con cinco días anteriores, objetivo $t+1$:

edad	Prod(t-5)	Prod(t-4)	Prod(t-3)	Prod(t-2)	Prod(t-1)	Prod(t+1)
0.725978648	0.910039856	0.89732397	0.899221864	0.893338394	0.902068704	0.901499336
0.754448399	0.884987664	0.882710192	0.88308977	0.890681344	0.89466692	0.890681344
0.277580071	0.953311824	0.951034352	0.947618144	0.944771304	0.936610362	0.941924464

Ilustración 33: Patrones problema n=5 y objetivo t+1.

- Patrón con tres días anteriores, objetivo $t + 2$:

edad	Prod(t-3)	Prod(t-2)	Prod(t-1)	Prod(t+2)
0.113879004	0.799582463	0.819510344	0.830897704	0.852723477
0.857651246	0.916302904	0.916872272	0.907762384	0.904915544
0.28113879	0.951034352	0.947618144	0.944771304	0.936610362

Ilustración 34: Patrones problema n=3 y objetivo t+2.

- Patrón con cuatro días anteriores, objetivo $t + 2$:

edad	Prod(t-4)	Prod(t-3)	Prod(t-2)	Prod(t-1)	Prod(t+2)
0.288256228	0.944771304	0.936610362	0.942304043	0.941924464	0.937559309
0.953736655	0.89447713	0.896375024	0.895236288	0.884987664	0.890681344
0.715302491	0.910609224	0.907762384	0.90871133	0.910039856	0.89732397

Ilustración 35: Patrones problema n=4 y objetivo t+2.

- Patrón con cinco días anteriores, objetivo $t + 2$:

edad	Prod(t-5)	Prod(t-4)	Prod(t-3)	Prod(t-2)	Prod(t-1)	Prod(t+2)
0.462633452	0.910609224	0.915733536	0.919719112	0.90890112	0.910039856	0.9145948
0.580071174	0.911178592	0.916302904	0.915733536	0.909470488	0.908141962	0.90871133
0.644128114	0.884987664	0.890681344	0.902068704	0.899032074	0.902068704	0.88897324

Ilustración 36: Patrones problema n=5 y objetivo t+2.

5.4.3. Parte I: predicción de $t + 1$

El primer objetivo, como se indicó en la definición del mismo, es la predicción de $t + 1$. Para ello, en la sección anterior se generaron previamente los conjuntos de datos que se van a utilizar, pero antes de proceder a la resolución del problema es necesario definir las arquitecturas de red que se van a utilizar y el procedimiento que se va a seguir para resolverlo.

Debido a que en el presente problema no es necesario únicamente averiguar la mejor arquitectura de red, sino que también es necesario averiguar el número de días anteriores o entradas que minimizan el error, se ha decidido utilizar la misma configuración de capas ocultas para los tres tipos de patrones de datos y comparar el mejor resultado para cada tipo de patrón para así obtener la mejor solución posible a la par que se averigua el mejor número de entradas al problema.

Las arquitecturas de red que a utilizar para resolver el problema son las siguientes:

- Arquitectura 1:
  - Capa de entrada: 3, 4 ó 5 neuronas dependiendo del fichero de datos.
  - Capa oculta 1: 1 neurona.
  - Capa de salida: 1 neurona.
- Arquitectura 2:
  - Capa de entrada: 3, 4 ó 5 neuronas dependiendo del fichero de datos.
  - Capa oculta 1: 5 neuronas.
  - Capa de salida: 1 neurona.
- Arquitectura 3:
  - Capa de entrada: 3, 4 ó 5 neuronas dependiendo del fichero de datos.
  - Capa oculta 1: 10 neuronas.
  - Capa de salida: 1 neurona.
- Arquitectura 4:
  - Capa de entrada: 3, 4 ó 5 neuronas dependiendo del fichero de datos.
  - Capa oculta 1: 5 neuronas.
  - Capa oculta 2:4 neuronas.
  - Capa de salida: 1 neurona.

Para todas las pruebas se ha decidido utilizar la función de activación lineal para las neuronas de la capa de salida, para así tratar de evitar que se produzca saturación, y se puedan predecir valores de producción correspondientes a los valores mínimo y máximo. Como tasas de aprendizaje se ha decidido utilizar 0.05, 0.1 y 0.2.

Por último, al contrario que en las pruebas de validación que se utilizó un número constante de iteraciones, en este problema se ha decidido utilizar las iteraciones necesarias para minimizar el error en cada prueba.

En primer lugar se estudia la solución para los datos que utilizan como entrada tres días pasados:

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 1	0.05	14000	$8.693161E^{-5}$	$1.141367E^{-4}$
Arquitectura 1	0.1	11000	$8.330847E^{-5}$	$1.139190E^{-4}$
Arquitectura 1	0.2	7000	$8.411703E^{-5}$	$1.130905E^{-4}$
Arquitectura 2	0.05	11000	$8.643396E^{-5}$	$1.161122E^{-4}$
Arquitectura 2	0.1	10000	$8.155623\ E^{-5}$	$1.158452E^{-4}$
Arquitectura 2	0.2	8000	$8.113190E^{-5}$	$1.151278E^{-4}$
Arquitectura 3	0.05	12000	$8.309740E^{-5}$	$1.173358E^{-4}$
Arquitectura 3	0.1	10000	$8.480727E^{-5}$	$1.116412E^{-4}$

Arquitectura 3	0.2	9000	$7.924347E^{-5}$	$1.174665E^{-4}$
Arquitectura 4	0.05	12000	$9.070486E^{-5}$	$1.291565E^{-4}$
Arquitectura 4	0.1	13000	$9.257486E^{-5}$	$1.332773E^{-4}$
Arquitectura 4	0.2	12000	$8.375150E^{-5}$	$1.182127E^{-4}$

Tabla 104: Resultados utilizando 3 días anteriores y objetivo t+1.

Una vez realizadas las pruebas, se puede observar que la mejor solución obtenida, al tener el menor error medio entre entrenamiento y test es la arquitectura 3 utilizando como razón de aprendizaje 0.2.

En este conjunto de experimentos se puede observar, como sucedía en las pruebas de la validación del simulador, que a menor razón de aprendizaje el algoritmo por lo general el algoritmo tarda más en converger en una solución. Por otra parte, se puede apreciar como la arquitectura 1, con una única capa oculta, y la arquitectura 4 con dos capas ocultas obtienen peores resultados que las arquitecturas 2 y 3, si bien la diferencia no es muy importante.

A continuación se realizan las pruebas para los siguientes ficheros de datos, aquellos correspondientes a los patrones de datos que utilizan los cuatro días anteriores como entrada.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 1	0.05	14000	$6.804222E^{-5}$	$1.1623737E^{-4}$
Arquitectura 1	0.1	11000	$6.759797E^{-5}$	$1.2012193E^{-4}$
Arquitectura 1	0.2	10000	$7.103442E^{-5}$	$1.2348894E^{-4}$
Arquitectura 2	0.05	15000	$6.702732E^{-5}$	$1.1833973E^{-4}$
Arquitectura 2	0.1	9000	$6.781490E^{-5}$	$1.2142744E^{-4}$
Arquitectura 2	0.2	9000	$7.379634E^{-5}$	$1.2815034E^{-4}$
Arquitectura 3	0.05	15000	$6.415731E^{-5}$	$1.1268965E^{-4}$
Arquitectura 3	0.1	10000	$7.030656E^{-5}$	$1.2614865E^{-4}$
Arquitectura 3	0.2	8000	$6.742866E^{-5}$	$1.1559626E^{-4}$
Arquitectura 4	0.05	15000	$7.164265E^{-5}$	$1.1762556E^{-4}$
Arquitectura 4	0.1	9000	$7.110163E^{-5}$	$1.2812559E^{-4}$
Arquitectura 4	0.2	8000	$7.385824E^{-5}$	$1.2945405E^{-4}$

Tabla 105: Resultados utilizando 4 días anteriores y objetivo t+1.

Una vez finalizadas las pruebas se puede observar que el mejor resultado se obtiene con la arquitectura 3 utilizando razón de aprendizaje de 0.05. Al igual que sucedía en los experimentos anteriores, se puede observar como una mayor razón de aprendizaje ralentiza la convergencia y como la arquitectura 4 obtienen peores resultados que las demás.

Por último, se realizan los experimentos para los datos que utilizan los cinco días anteriores para calcular $t + 1$.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 1	0.05	13000	$8.297032E^{-5}$	$1.097691E^{-4}$
Arquitectura 1	0.1	12000	$8.984483E^{-5}$	$1.183375E^{-4}$
Arquitectura 1	0.2	10000	$1.046727E^{-4}$	$1.355182E^{-4}$
Arquitectura 2	0.05	15000	$8.392264E^{-5}$	$1.145827E^{-4}$
Arquitectura 2	0.1	12000	$8.766993E^{-5}$	$1.125426E^{-4}$
Arquitectura 2	0.2	10000	$1.098579E^{-4}$	$1.412955E^{-4}$
Arquitectura 3	0.05	15000	$9.552774E^{-5}$	$1.298296E^{-4}$
Arquitectura 3	0.1	13000	$1.063977E^{-4}$	$1.369902E^{-4}$
Arquitectura 3	0.2	11000	$1.044072E^{-4}$	$1.334213E^{-4}$
Arquitectura 4	0.05	15000	$8.853642E^{-5}$	$1.167895E^{-4}$
Arquitectura 4	0.1	14000	$1.013307E^{-4}$	$1.337896E^{-4}$
Arquitectura 4	0.2	13000	$1.001736\ E^{-4}$	$1.299057E^{-4}$

Tabla 106: Resultados utilizando 5 días anteriores y objetivo t+1.

Una vez finalizada la prueba se puede observar que el mejor resultado obtenido es para la arquitectura 1 utilizando como razón de aprendizaje 0.05.

En la tabla 107 se puede comparar la mejor solución obtenida para cada uno de los ficheros de datos, pudiendo así determinar la mejor solución obtenida y el número de días anteriores o entradas que obtiene mejores resultados.

Días anteriores	Arquitectura	Tasa de aprendizaje	Error de entrenamiento	Error de test
3	Arquitectura 3	0.2	$7.924347E^{-5}$	$1.174665E^{-4}$
4	Arquitectura 3	0.05	$6.415731E^{-5}$	$1.1268965E^{-4}$
5	Arquitectura 1	0.05	$8.297032E^{-5}$	$1.097691E^{-4}$

Tabla 107: Comparativa de resultados objetivo t+1.

Comparando los errores obtenidos se puede comprobar que la mejor solución obtenida es aquella que utiliza los patrones de datos que tienen como entrada los 4 días anteriores, aunque observando el error de test, la mejor solución se obtendría con 5 valores anteriores. A continuación se mostrará una gráfica comparativa entre las salidas obtenidas y las salidas deseadas para entrenamiento y test (ilustración 37 e ilustración 38) para dicha solución. Para realizar las gráficas primero se des-normalizaran las salidas, para ver así el número real de huevos.

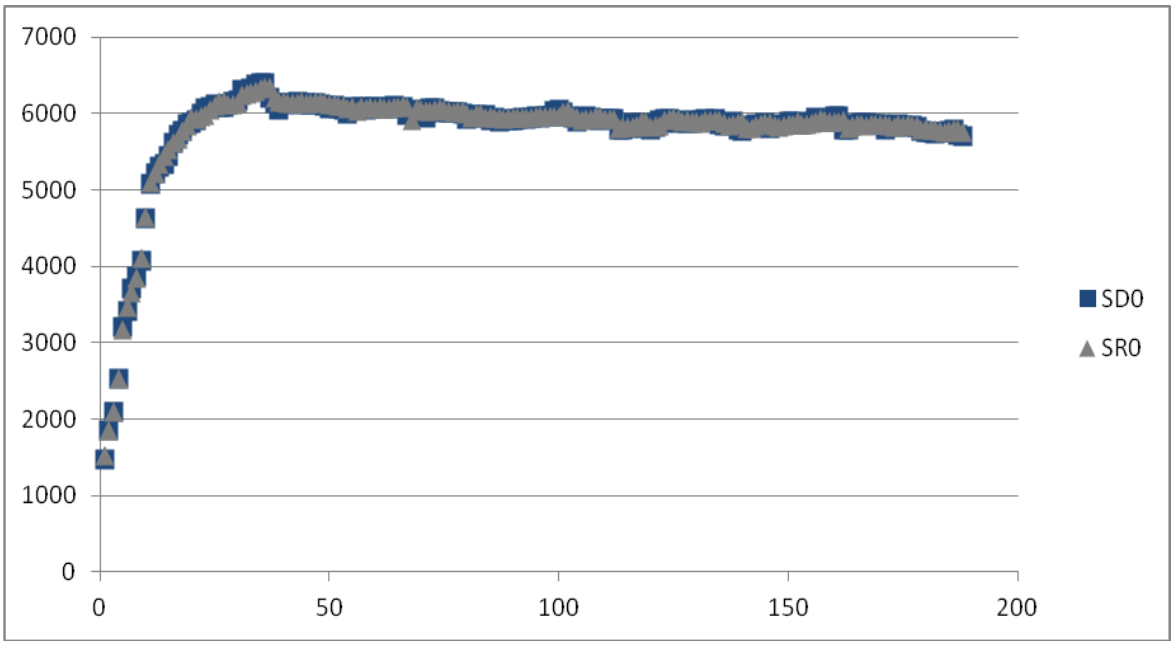

Ilustración 37: Comparativa de salidas deseadas y obtenidas para t+1 (entrenamiento).

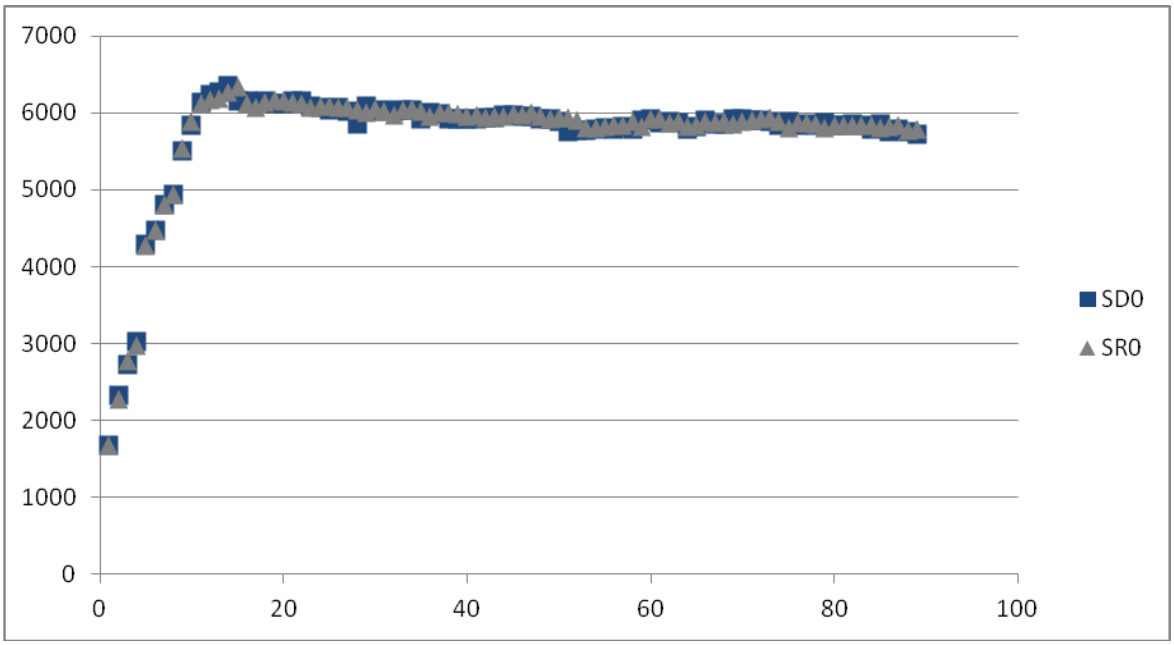

Ilustración 38: Comparativa de salidas deseadas y obtenidas para t+1 (test).

Por último se muestra en un gráfico la unión de los datos de entrenamiento y test que en su conjunto representa las predicciones de la producción de huevos para la vida completa del lote de gallinas en la granja. También se incluye la predicción del modelo de LokHorst para poder así comprobar como el Perceptrón Multicapa predice las oscilaciones en la producción.

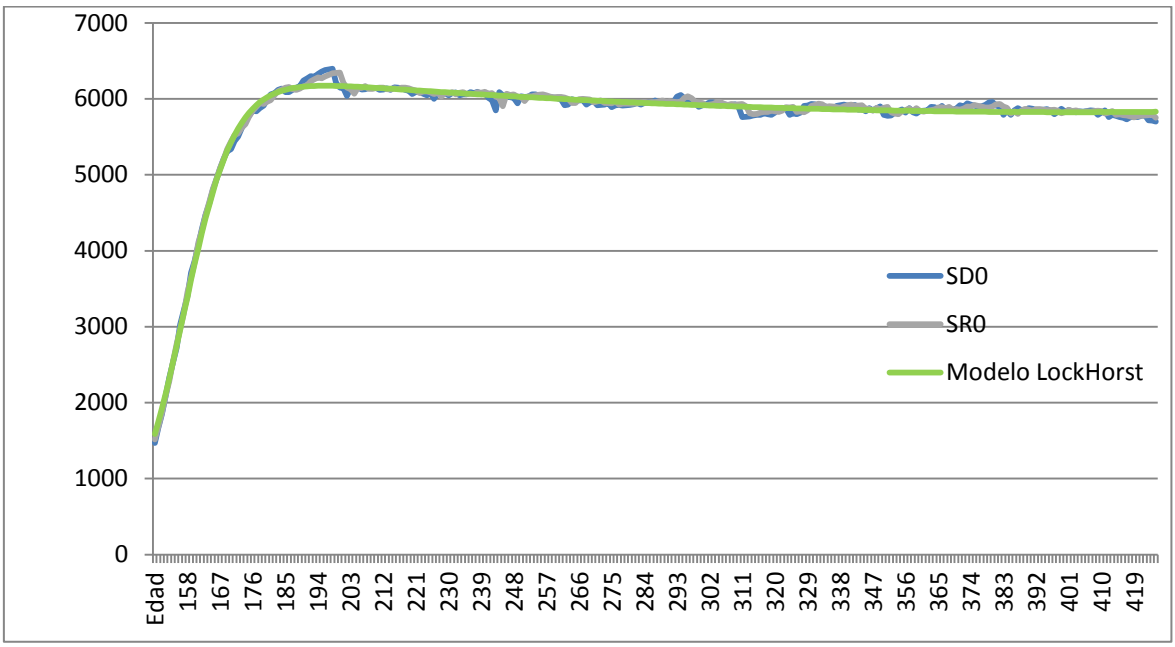

Ilustración 39: Comparativa entre salidas deseadas y obtenidas para t+1 (total).

En la ilustración anterior se puede observar como la solución obtenida aproxima las oscilaciones en la producción de huevos mientras que el modelo de LokHorst únicamente predice la tendencia de la producción.

En la siguiente tabla se puede consultar el error medio en número de huevos al día, que se produce diariamente en la predicción de la producción en el instante de tiempo $t+1$ utilizando la arquitectura obtenida y el modelo matemático.

Patrones	Error de la solución obtenida	Error modelo de LokHorst
Entrenamiento	31.5611022	42.35914884
Test	40,722716	34,1576006
Entrenamiento + Test	34.5047254	43.5038627

Tabla 108: Error medio obtenido en la predicción de t+1.

5.4.4. Parte II: predicción de $t+2$

Una vez satisfecho el primer objetivo del problema, el segundo objetivo, como se comentó anteriormente, es la predicción de la producción de huevos en $t+2$. Para ello una vez más es necesario determinar tanto la arquitectura como el número de entradas que mejor se adapta al problema minimizando así el error.

Para la predicción de $t+2$, y con el objetivo de simplificar los experimentos, solo se van a llevar a cabo pruebas utilizando las arquitecturas 2 (5 neuronas ocultas) y 3 (10 neuronas ocultas), pues en términos generales son las que proporcionaron mejores o equivalentes resultados. Por otra parte, al igual que en el apartado anterior, se utilizará la función de activación lineal en la salida para evitar que se produzca saturación y se utilizarán como razón de aprendizaje 0.05, 0.1 y 0.2.

Al igual que en el apartado anterior, se mostrarán los resultados para cada uno de los ficheros de datos en tablas diferentes y por último se comparará el mejor resultado obtenido para cada uno.

A continuación se muestran los resultados obtenidos utilizando los ficheros de datos que utilizan 3 días anteriores.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 2	0.05	15000	$1.049021E^{-4}$	$1.477072E^{-4}$
Arquitectura 2	0.1	13000	$1.148555E^{-4}$	$1.665532E^{-4}$
Arquitectura 2	0.2	12000	$1.302259E^{-4}$	$1.926971E^{-4}$
Arquitectura 3	0.05	14000	$1.297008E^{-4}$	$1.718652E^{-4}$
Arquitectura 3	0.1	14000	$1.187381E^{-4}$	$1.644425E^{-4}$
Arquitectura 3	0.2	12000	$1.179237E^{-4}$	$1.738943E^{-4}$

Tabla 109: Resultados utilizando 3 días anteriores y objetivo t+2.

Una vez realizadas las pruebas se puede observar como una menor razón de aprendizaje conlleva una convergencia más lenta del algoritmo de aprendizaje. Por otra parte, se puede apreciar como la predicción de $t + 2$ obtiene peores resultados que la predicción de $t + 1$, lo cual resulta razonable al desconocer los datos de un mayor intervalo de días entre los conocidos y el que se desea predecir.

Se puede observar que la mejor solución utilizando tres días anteriores es la arquitectura 2 utilizando la razón de aprendizaje 0.05 al tener el menor error tanto en entrenamiento como en test.

A continuación se muestran los resultados obtenidos utilizando los ficheros de datos que utilizan 4 días anteriores.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 2	0.05	15000	0.0040664	$3.016387E^{-4}$
Arquitectura 2	0.1	12000	0.0039842	$3.732462E^{-4}$
Arquitectura 2	0.2	11000	0.0038935	$3.760959E^{-4}$
Arquitectura 3	0.05	12000	0.0040495	$2.934883E^{-4}$
Arquitectura 3	0.1	11000	0.0039969	$3.153422E^{-4}$
Arquitectura 3	0.2	9000	0.0039986	$3.188840E^{-4}$

Tabla 110: Resultados utilizando 4 días anteriores y objetivo t+2.

El mejor resultado obtenido por su media de error de entrenamiento y test se puede ver que es el obtenido por la arquitectura 3 utilizando razón de aprendizaje 0.2.

Por último se muestran los resultados obtenidos utilizándolos ficheros de datos que utilizan 5 días anteriores.

Arquitectura	Tasa de aprendizaje	Iteraciones	Error de entrenamiento	Error de test
Arquitectura 2	0.05	21000	$8.894752E^{-5}$	$1.185101E^{-4}$
Arquitectura 2	0.1	18000	$8.847851E^{-5}$	$1.221799E^{-4}$
Arquitectura 2	0.2	18000	$8.470382E^{-5}$	$1.182270E^{-4}$
Arquitectura 3	0.05	21000	$8.996992E^{-5}$	$1.227605E^{-4}$
Arquitectura 3	0.1	16000	$8.282160E^{-5}$	$1.130442E^{-4}$
Arquitectura 3	0.2	15000	$8.364437E^{-5}$	$1.207295E^{-4}$

Tabla 111: Resultados utilizando 5 días anteriores y objetivo t+2.

Se puede observar que el mejor resultado es el obtenido por la arquitectura 3 utilizando 0.1 como razón de aprendizaje. A continuación en la tabla 111 se muestran los mejores resultados para cada unos de los ficheros de prueba:

Días anteriores	Arquitectura	Tasa de aprendizaje	Error de entrenamiento	Error de test
3	Arquitectura 2	0.05	$1.049021E^{-4}$	$1.477072E^{-4}$
4	Arquitectura 3	0.2	0.0039986	$3.188840E^{-4}$
5	Arquitectura 3	0.1	$8.282160E^{-5}$	$1.130442E^{-4}$

Tabla 112: Comparativa de resultados objetivo t+2.

Comparando los errores obtenidos se puede comprobar que la mejor solución obtenida es aquella que utiliza los patrones de datos que tienen como entrada los 5 días anteriores. A continuación se mostrará una gráfica comparativa entre las salidas obtenidas y las salidas deseadas para entrenamiento y test (ilustración 40 e ilustración 41) para dicha solución. Al igual que en el apartado anterior se des-normalizaran las salidas.

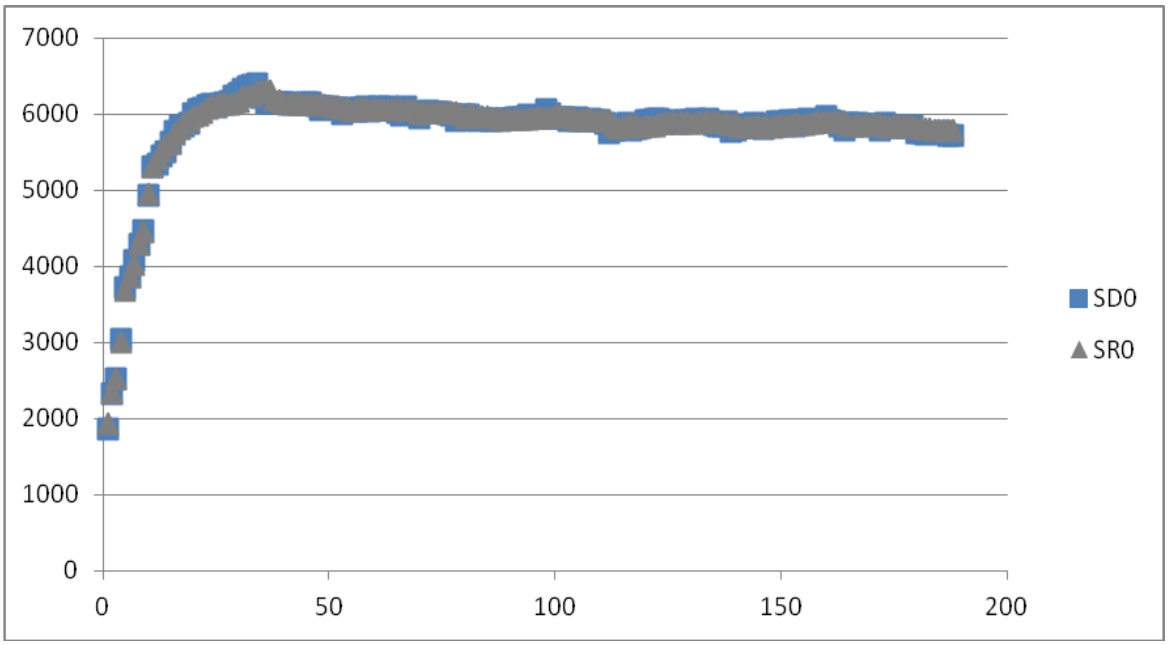

Ilustración 40: Comparativa de salidas deseadas y obtenidas para t+2 (entrenamiento).

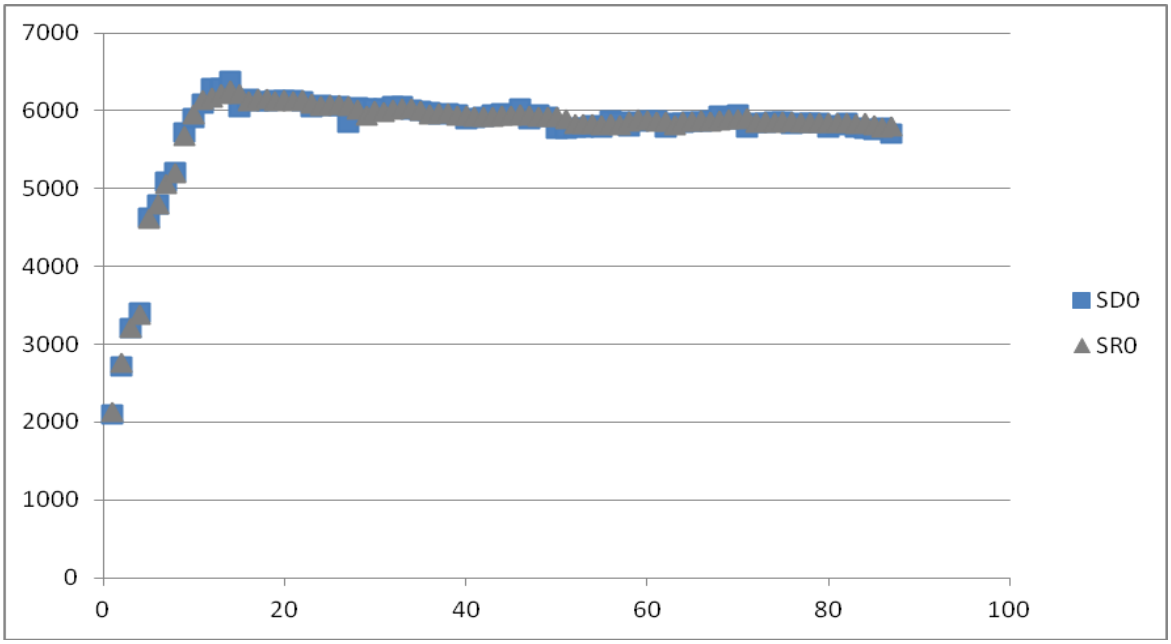

Ilustración 41: Comparativa de salidas deseadas y obtenidas para t+2 (test).

Por último se muestra en un gráfico la unión de los datos de entrenamiento y test que en su conjunto representa las predicciones de la producción de huevos para la vida completa del lote de gallinas en la granja, así como de las salidas predichas por el modelo LokHorst.

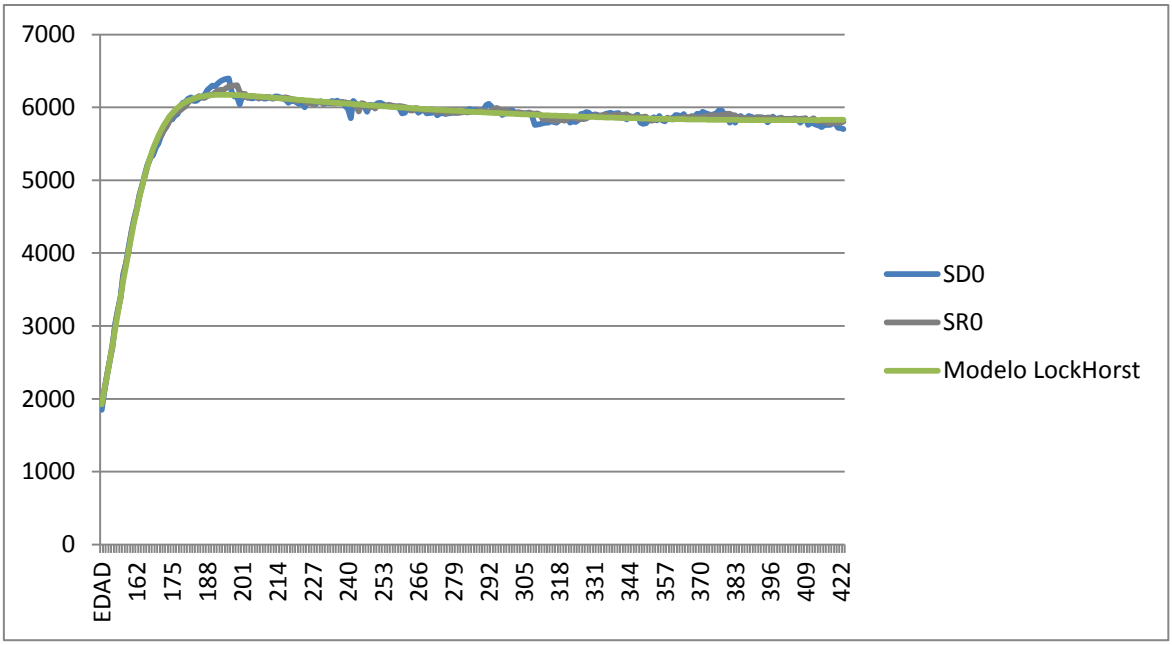

Ilustración 42: Comparativa entre salidas deseadas y obtenidas para t+2 (total).

En la siguiente tabla se muestra el error medio en número de huevos por día cuando el horizonte de predicción es $t + 2$ utilizando la arquitectura obtenida y el modelo matemático.

Patrones	Error de la solución obtenida	Error modelo de LokHorst
Entrenamiento	36.0937786	42.92212763
Test	38.7974078	43.51114913
Entrenamiento + Test	36.9491086	43.10847261

Tabla 113: Error medio obtenido en la predicción de t+2.

Una vez obtenidas las soluciones para la predicción de $t + 1$ y $t + 2$ se pueden obtener diversas conclusiones. Observando los resultados se puede comprobar que el error obtenido en ambas predicciones es menor para aquellos patrones que se utilizaron como de conjunto de entrenamiento, este es un suceso normal cuando se trabaja con redes de neuronas, dado que los pesos de la red se adaptan en función de dichos patrones.

También se puede observar que se obtiene un mejor resultado para la predicción en $t + 1$ que en $t + 2$, esta observación se explica en el mayor periodo de incertidumbre, es decir, al desconocimiento de un mayor número de datos. La peculiaridad de que el error sobre los patrones del conjunto de test en la predicción de $t + 2$ sea menor que en $t + 1$ puede deber a las arquitecturas seleccionadas.

Tras estudiar los resultados obtenidos se puede concluir que el Perceptrón Multicapa es capaz de obtener resultados aceptables tanto en la predicción de $t + 1$ y $t + 2$.

Capítulo 6. Manual de Usuario

6.1. Introducción

Bienvenido al manual de usuario del Simulador Del Perceptrón Multicapa (SPM) A lo largo de las siguientes páginas se describirá de manera clara y concisa la información necesaria para aprender a utilizar el simulador SPM. Con el fin de facilitar la comprensión del manual, se incluyen imágenes explicativas.

Antes de comenzar a utilizar el simulador como usuario no será necesario realizar ninguna instalación, bastará con mantener la estructura de archivos del programa para su correcto funcionamiento.

6.2. Ejecución del simulador

El simulador así como todos sus archivos se encuentran dentro de una carpeta bajo el nombre SPM. Esta carpeta contiene todo lo necesario para poder ejecutar el simulador, la carpeta contiene los siguientes ficheros y subcarpetas:

- SPM.jar: es un fichero ejecutable que lanza el simulador para que el usuario pueda comenzar a trabajar.
- Imágenes: la carpeta Imágenes contiene imágenes que utiliza el simulador durante su ejecución.
- Datos: la carpeta Datos contiene distintos ejemplos de ficheros de datos con el formato que requiere el simulador ya aplicado.
- Redes: la carpeta Redes contiene distintos ejemplos de ficheros de arquitecturas de red guardadas por el simulador.
- Resultados: la carpeta Resultados contiene distintos ejemplos de ficheros de resultados guardados por el simulador.

Para poder utilizar el simulador el usuario deberá ejecutar el fichero SPM.jar, al hacerlo le aparecerá la siguiente ventana de bienvenida (ilustración 43):

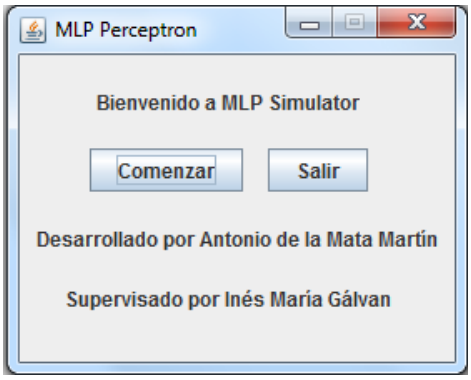

Ilustración 43: Ventana de bienvenida.

Para poder empezar a utilizar el simulador el usuario deberá pulsar el botón **Comenzar** lo cual le llevará a la siguiente ventana (ilustración 44) donde ya podrá hacer uso de las distintas funcionalidades que ofrece el simulador.

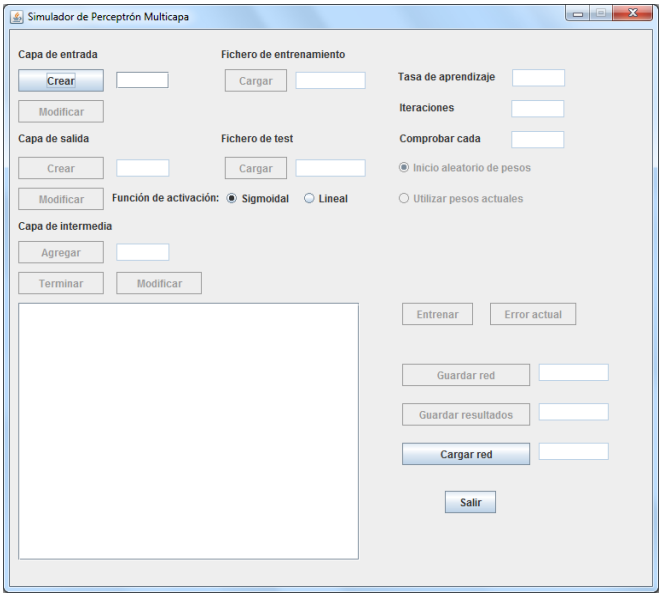

Ilustración 44: Interfaz del simulador.

6.3. Ficheros de entrenamiento y test

A lo largo de esta sección, el usuario podrá leer recomendaciones para el pre-procesado de datos y cómo ha de preparar los ficheros de datos para su compatibilidad con el simulador SPM.

6.3.1. Pre-procesado de datos

Una vez el usuario cuente con los datos del dominio sobre los que va a realizar las simulaciones, es recomendable que se lleve a cabo un pre-procesado de los mismos, facilitando de esta manera la obtención de mejores resultados por parte del simulador.

Este pre-procesado de los datos está compuesto por dos fases llamados Normalización y Aleatorización.

6.3.1.1. Normalización

En caso de que los datos de entrada no estén normalizados entre 0 y 1, es decir que no todos los datos tengan un valor entre 0 y 1, es recomendable, aunque no imprescindible, normalizar los atributos del fichero de datos para conseguir así unos mejores resultados.

Uno de los motivos de hacer esto es darle a todos los atributos del fichero de datos la misma importancia al comienzo de la simulación, evitando así que atributos con diferentes escalas tengan diferente relevancia al calcular la salida.

Para llevar a cabo el proceso de normalización el usuario deberá normalizar uno a uno cada uno de los atributos de entrada, siendo recomendable también normalizar las salidas.

Para ello, se deben calcular los valores máximos y mínimos de cada atributo. La expresión que resume el proceso de normalización para los distintos valores de un atributo es la siguiente:

$$X^n = \frac{X^p - min}{max - min}$$

Ecuación 17: Normalización de datos.

donde X^n es el valor normalizado, X^pel valor original, min es el valor mínimo del atributo que se está normalizando y max es el valor máximo para el atributo que se está normalizando.

Si se normalizan las salidas, será necesario que el usuario haga el proceso inverso o de des-normalización de las salidas, para así poder ver los resultados en su escala original. Para ello deberá utilizar los máximos y los mínimos de las salidas originales para calcularlo. La expresión que resume el proceso de des-normalizado, viene de despejar el valor X^p en la ecuación anterior:

$$X^p = X^n * (max\ - min) + min$$

Ecuación 18: Des-normalización de datos.

Donde X^n es el valor normalizado obtenido por el simulador, X^pel valor del atributo desnormalizado, min es el valor mínimo de la salida antes de normalizarla y max es el valor máximo de la salida antes de normalizarla.

6.3.1.2. Aleatorización

La aleatorización de los datos es un procedimiento que es recomendable realizar por parte del usuario, especialmente cuando se parte de un único conjunto de datos que se va a subdividir en dos ficheros que serán utilizados de manera independiente como ficheros de test y entrenamiento.

El proceso de aleatorización ayudará a evitar que casos relevantes que puede que se encuentren muy cercanos en el conjunto de datos queden representados únicamente en uno de los ficheros de datos.

Para llevar a cabo la aleatorización el usuario deberá simplemente cambiar el orden de los patrones de modo que estén ordenados de manera aleatoria. Para ello el usuario podrá utilizar diferentes herramientas, como por ejemplo Microsoft Excel.

6.3.2. Formato de los ficheros y cabeceras

Es necesario que los ficheros de datos que se vayan a utilizar para llevar a cabo las simulaciones tengan un formato específico para que el simulador pueda interpretar los datos correctamente.

La estructura que deberán tener los ficheros de datos es la siguiente:

- Se utiliza el carácter "." como separador decimal.
- En la primera línea de cada fichero el usuario tendrá que escribir una cabecera compuesta por los tres siguientes valores: total de atributos, número de entradas y número de salidas.
- Cada patrón deberá estar en una línea distinta, el primero de ellos deberá estar situado en la segunda línea dado que la primera línea se reserva para la cabecera anteriormente mencionada.
- Dentro de un mismo fichero, todos los datos tanto de la cabecera como de los patrones deberán estar separados entre sí ya sea con espacios o con tabulaciones.
- Es necesario dejar una línea en blanco después del último patrón.

A continuación se muestran dos ejemplos de formato de fichero de entrada (ilustración 45 e ilustración 46):

- Ejemplo 1: Patrones de datos de 4 entradas y 3 salidas, separado por espacios:

```
7 4 3
0.8 0.6 0.4 0.6 1 0 0
0.8 1 1 0.4 1 0 0
0.8 0.2 0.6 0.2 1 0 0
0.8 1 0.4 0.8 1 0 0
1 0.2 0.6 1 0 0 1
0.2 1 0.6 0.4 0 0 1
0.6 1 0.4 1 1 0 0
```

Ilustración 45: Ejemplo de fichero de datos 1.

- Ejemplo 2: Patrones de datos de 4 entradas y 3 salida, separado por tabulaciones:

```
7   4   3
0.8 0.6 0.4 0.6 1   0   0
0.8 1   1   0.4 1   0   0
0.8 0.2 0.6 0.2 1   0   0
0.8 1   0.4 0.8 1   0   0
1   0.2 0.6 1   0   0   1
0.2 1   0.6 0.4 0   0   1
0.6 1   0.4 1   1   0   0
```

Ilustración 46: Ejemplo de fichero de datos 2.

6.3.3. Cargar ficheros de entrenamiento y test

El simulador cuenta con dos botones para cargar los ficheros de datos, uno para entrenamiento y uno para test, estos botones están deshabilitados al iniciar el simulador y se habilitarán de manera automática cuando sea posible cargar los ficheros de datos.

Cuando se define una arquitectura o se carga una red se habilitan los botones para la carga de los ficheros de datos, en primer lugar se habilita el botón para la carga de datos de entrenamiento (ilustración 47).

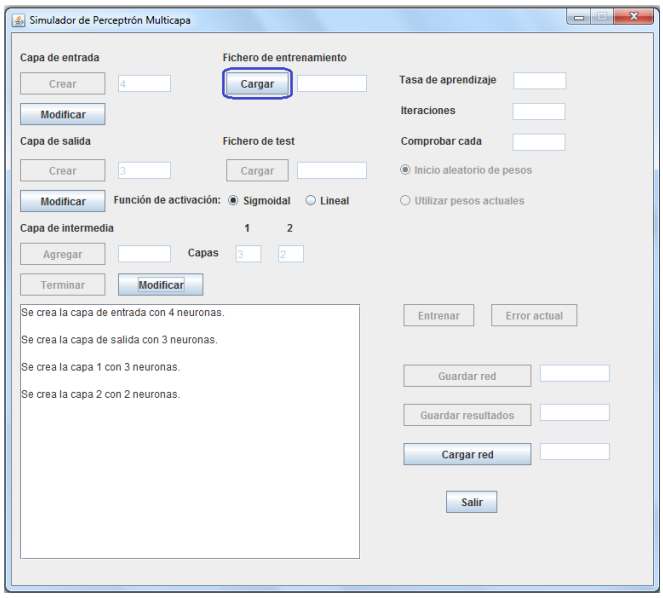

Ilustración 47: Carga de datos de entrenamiento habilitada.

Para cargar los datos una vez el simulador haya habilitado la carga, simplemente tendremos que pulsar el botón de **Cargar** y nos aparecerá una ventana para la búsqueda de ficheros (ilustración 48). La ventana de búsqueda viene por defecto con un filtro para ficheros para que muestre los ficheros con la extensión "txt" Si se han guardado los datos con una extensión diferente deberemos cambiar el filtro de la búsqueda en la propia ventana.

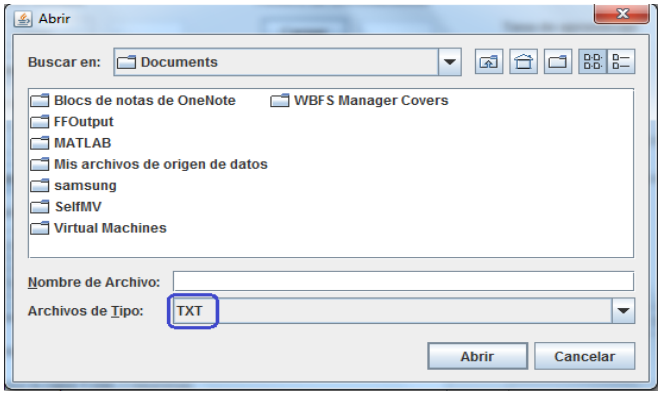

Ilustración 48: Ventana para la carga de ficheros de datos.

Una vez seleccionado el fichero de datos que se desee utilizar, el simulador muestra el resultado de la carga, tanto si la carga es correcta, como si no lo es. En caso de que la carga no se realizase correctamente el simulador no almacenaría ningún dato y nos devolvería un mensaje indicándonos la razón por la cual se ha producido el error.

A continuación se muestran los posibles casos derivados de la carga de un fichero de datos y las razones que pueden llevar a los mismos.

- Caso 1: La carga del fichero de datos se ha realizado correctamente (ilustración 49).

Cuando la carga del fichero de datos se ha realizado de manera correcta el simulador da una confirmación a través del cuadro de información de la esquina inferior izquierda y muestra el nombre del fichero actualmente cargado en el cuadro contiguo al botón de carga.

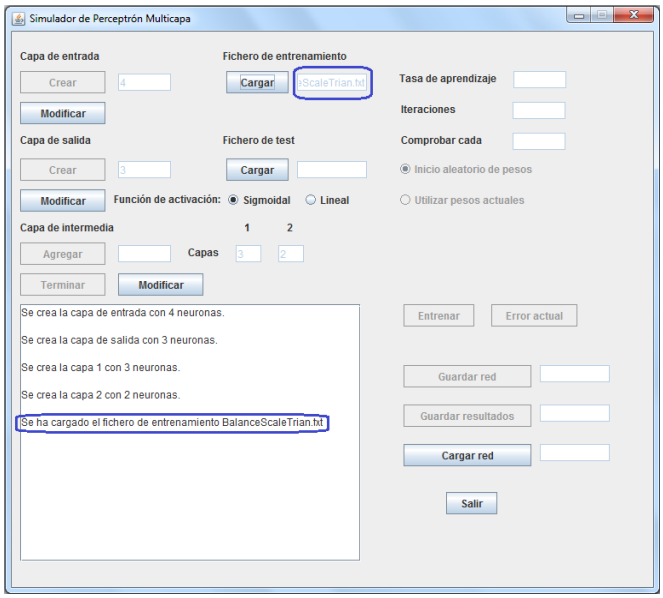

Ilustración 49: Carga de fichero de datos correcta.

- Caso 2: Errores en la cabecera del fichero de datos (ilustración 50 e ilustración 51).

  Cuando haya un error en la cabecera del fichero, el simulador devuelve dos posibles mensajes de error, el primero por la ausencia de la cabecera y el segundo porque los datos de ésta no concuerden.

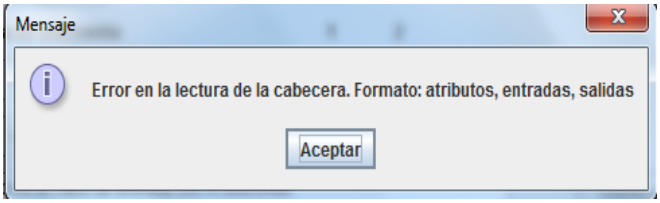

Ilustración 50: Error por ausencia de cabecera en los datos.

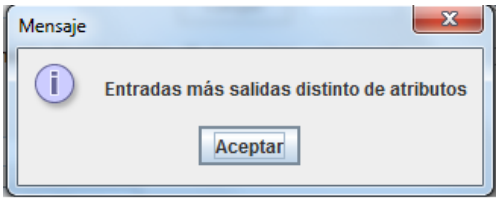

Ilustración 51: Error en los datos de la cabecera.

- Caso 3: Error en los patrones de datos (ilustración 52).

  Cuando el simulador devuelve es debido a los siguientes motivos:

  - Los patrones no concuerdan con la cabecera.
  - Que haya algún patrón en blanco en el fichero de datos.
  - Que haya algún patrón al que le falte un dato.
  - Que el separador decimal en los patrones sea la "," en lugar del ".".
  - Que falte un salto de línea tras el último patrón.

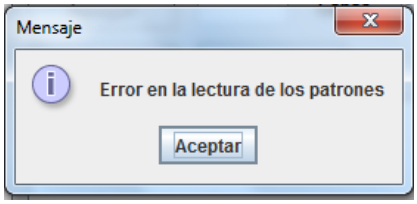

Ilustración 52: Error en los patrones de los ficheros de datos.

En caso de que al cargar los ficheros de datos suceda algún error el simulador lo indicará, además de con uno de los mensajes de error anteriormente mostrados, a través del cuadro de información de la esquina inferior izquierda (ilustración 53).

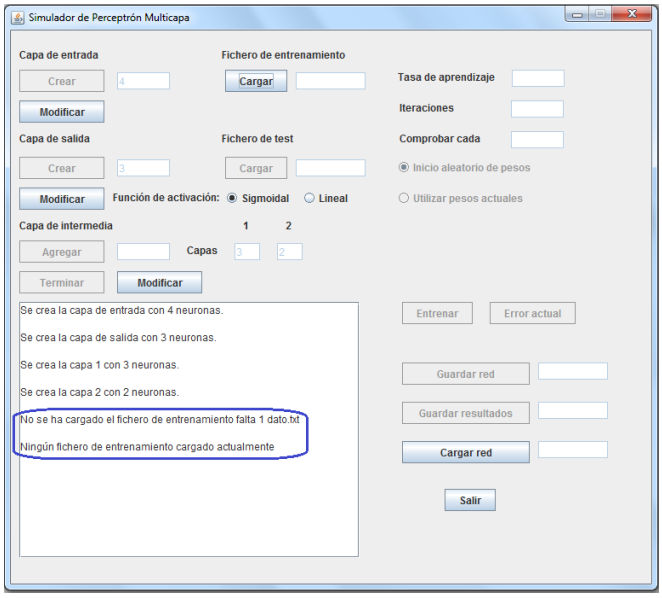

Ilustración 53: Confirmación de error en la carga de datos.

Este procedimiento para la carga de un fichero de datos para su uso como fichero de entrenamiento es igual para la carga de un fichero de datos que se vaya a utilizar como fichero de test.

En caso de que un usuario no desee utilizar dos ficheros (uno para entrenamiento y uno para test) porque cuente con pocos datos o por cualquier otro motivo, el usuario deberá cargar el mismo fichero de datos como entrenamiento y como test.

6.4. Definir una arquitectura de red

Para realizar una simulación, en primer lugar el usuario debe definir una arquitectura de red, la cual podrá ser modificada en todo momento para permitir al usuario probar diferentes arquitecturas a lo largo de las simulaciones.

Para definir la arquitectura de red el usuario podrá optar por dos posibilidades: definir una nueva arquitectura desde cero o cargar una red ya existente que el usuario guardase en simulaciones anteriores.

6.4.1. Definir una nueva arquitectura

Si el usuario opta por definir una nueva arquitectura de red debe definir los parámetros que conforman la misma, las neuronas de entrada y salida y la configuración de las capas ocultas (el número de capas y las neuronas que tiene cada una).

El usuario comenzará definiendo el número de neuronas de entrada, el cual deberá coincidir con las entradas que tengan las instancias del fichero de datos. Para crear la capa de entrada introduciremos la cantidad de neuronas necesarias en el cuadro indicado y se pulsará el botón **Crear** como se indica a continuación (ilustración 54).

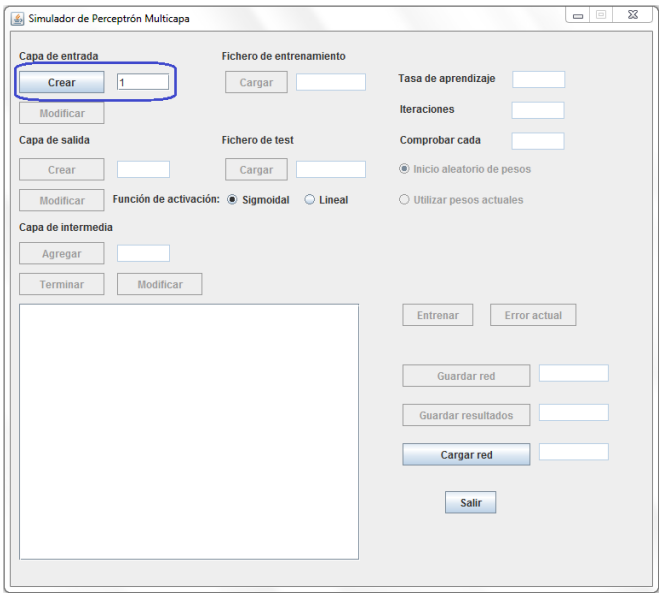

Ilustración 54: Definir arquitectura (capa de entrada).

Una vez definida la capa de entrada se habilitan los botones para modificarla y para crear la capa de salida (ilustración 55). Para definir la capa de salida, el usuario debe definir las neuronas de la capa de salida y pulsar el botón **Crear** correspondiente a la capa de salida. Las neuronas de salida de la arquitectura tendrán que coincidir con las salidas que se indiquen en los ficheros de datos que se van a utilizar para las simulaciones.

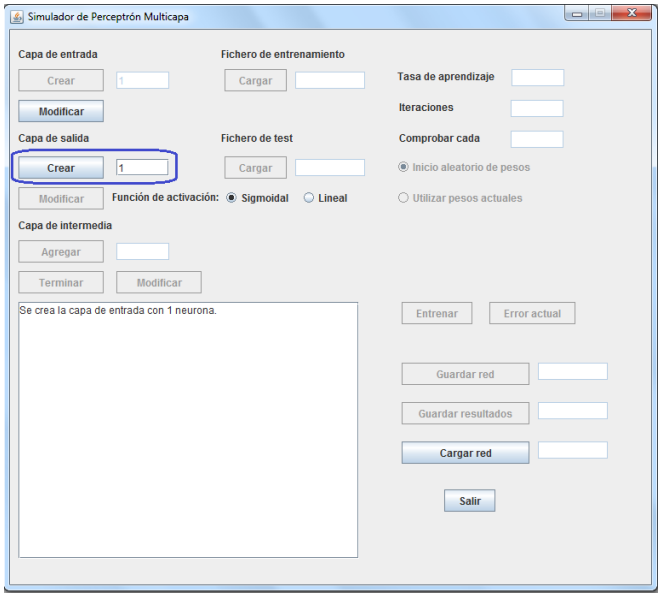

Ilustración 55: Definir arquitectura (capa de salida).

Una vez definida la capa de salida se habilitan los botones para modificarla y para crear las capas ocultas. En esta ocasión el usuario definirá las diferentes capas una a una, pulsando el botón **Agregar** para ir añadiéndolas de forma ordenada, siendo la primera capa la contigua a

las neuronas de entrada (se definen de izquierda a derecha). A medida que el usuario vaya agregando las diferentes capas en el simulador se hará visible la arquitectura actual. Una vez que el usuario finalice de agregar capas ocultas deberá pulsar el botón **Terminar** para poder así continuar con la simulación. El simulador SPM está diseñado para trabajar con un máximo de 10 capas ocultas.

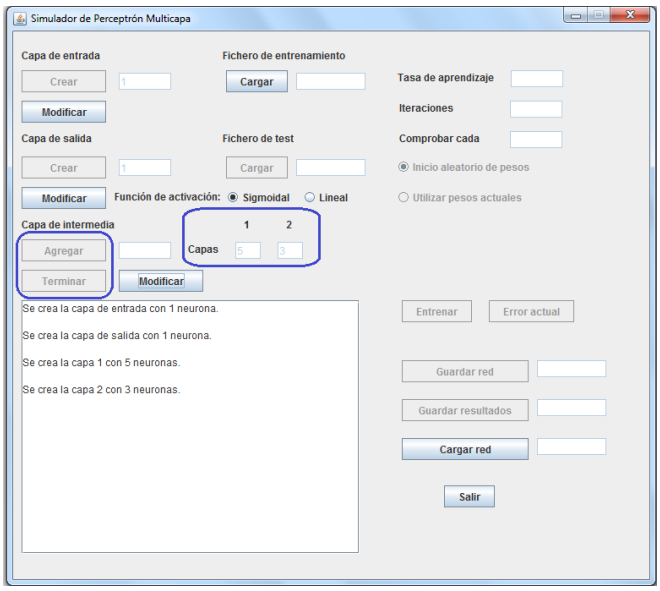

Ilustración 56: Definir arquitectura (capas ocultas).

Cada vez que se defina una capa, ya sea de entrada, oculta o de salida, el simulador mostrará una confirmación en la cuadro de información de la esquina inferior izquierda de que el procedimiento se ha realizado de manera correcta (ilustración 57).

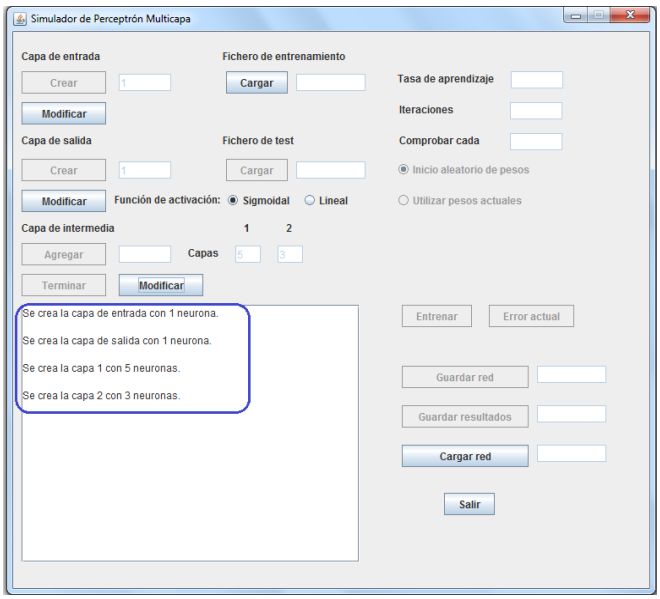

Ilustración 57: Definir arquitectura (confirmaciones).

Si al pulsar los botones de **Crear** o **Agregar** de las diferentes capas haya algún error en los datos introducidos, en lugar de esta confirmación en el cuadro de información, el simulador mostraría un mensaje indicando el error que se ha cometido (ilustración 58).

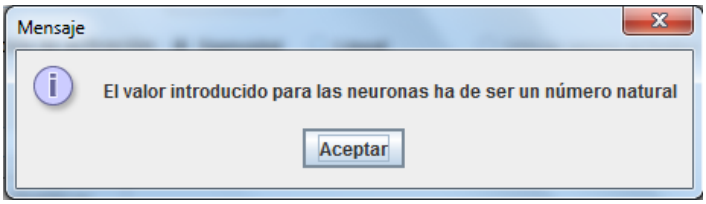

Ilustración 58: Definir una arquitectura (error).

En caso de que el usuario opte por cargar una arquitectura de red previamente definida y guardada, por favor consultar la sección 4.2, “Cargar una red”, de este manual.

6.4.2. Cargar una red

El usuario tendrá la opción de cargar una red que haya sido guardada en simulaciones anteriores realizadas con este simulador, para ello deberá pulsar el botón de **Cargar red** (ilustración 59); una vez lo haga le aparecerá una ventana para que busque y seleccione el fichero de red que desee cargar.

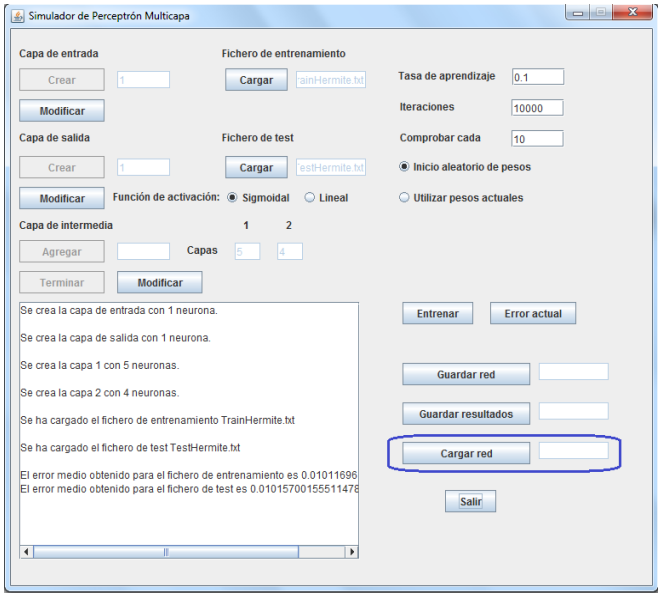

Ilustración 59: Cargar una red I.

Una vez seleccionado el fichero con la red, el simulador completa automáticamente los campos en los cuales se define la red. Además, el usuario recibirá confirmación de que la red

se ha cargado en el cuadro de información y se habilita el parámetro de inicialización de pesos en el entrenamiento y se habilitan las demás funcionalidades del simulador (ilustración 60).

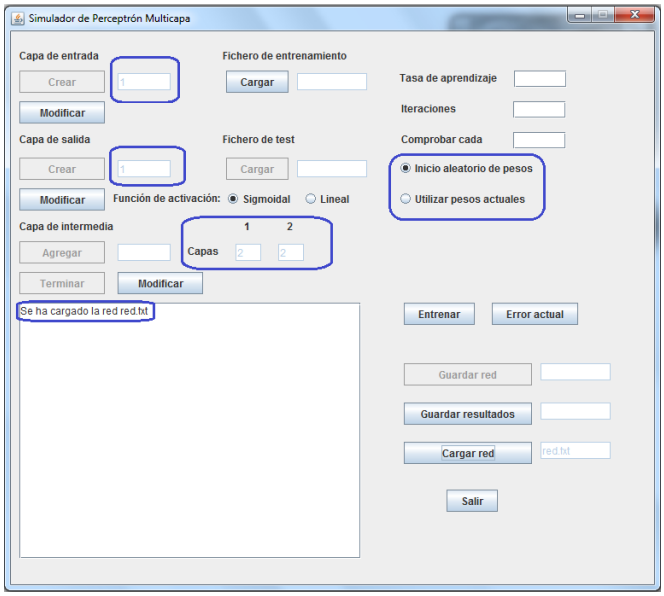

Ilustración 60: Cargar una red II.

6.4.3. Modificar una arquitectura

Si en algún momento el usuario decide modificar la arquitectura de red, podrá hacerlo siempre y cuando los botones de modificación estén habilitados (estos botones se deshabilitan mientras dura el proceso de entrenamiento o cuando no hay valores definidos). Para modificar la arquitectura de red el usuario podrá modificar la capa de entrada, de salida y las capas ocultas de manera independiente entre ellas (ilustración 61).

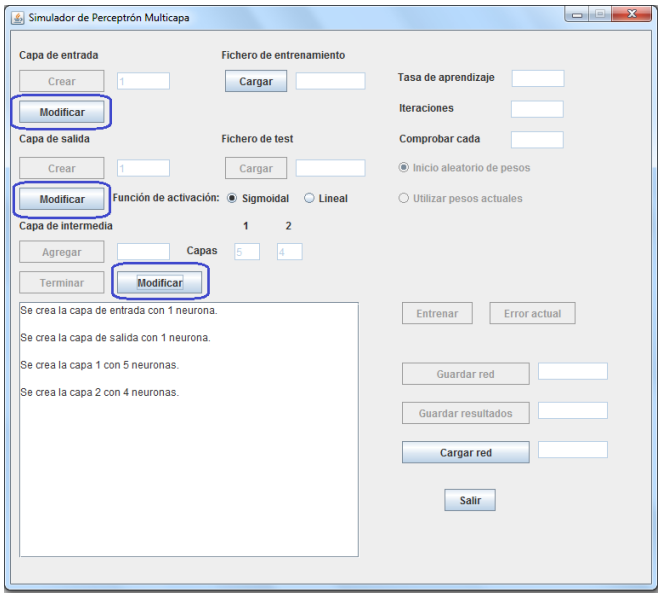

Ilustración 61: Modificar la arquitectura I.

A la hora de modificar la arquitectura de red, el simulador tiene dos protocolos distintos, uno para las capas de entrada y salida y otro para las capas ocultas.

Para modificar el numero de neuronas de entrada y salida el usuario debe pulsar su respectivo botón **Modificar**, insertar el nuevo valor deseado para el numero de neuronas y pulsar **Crear**. En caso de no pulsar dicho botón el simulador continuará utilizando el valor anterior (ilustración 62).

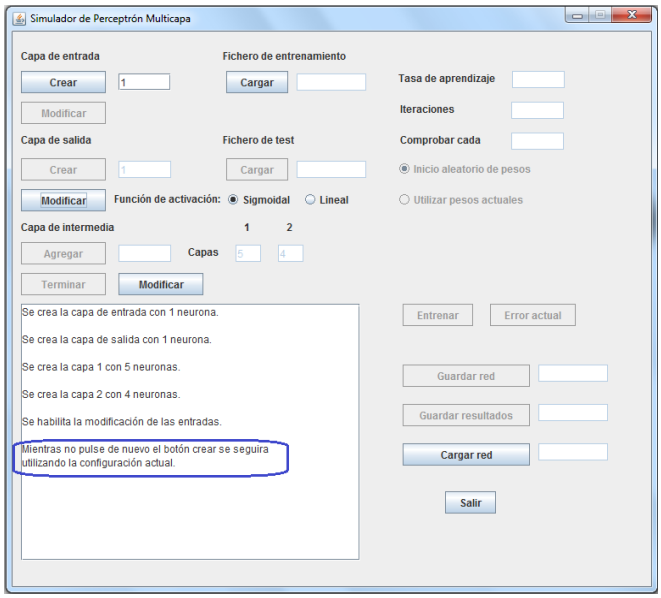

Ilustración 62: Modificar una arquitectura II.

La modificación de las capas ocultas funciona de manera diferente. Cuando el usuario pulse el botón para modificar la configuración de las capas ocultas aparece una ventana para que el

usuario pueda confirmar si desea modificar la configuración de las mismas (ilustración 63). En este mensaje se le advierte al usuario de que en caso de pulsar el botón de confirmación se borrará la configuración actual de las mismas.

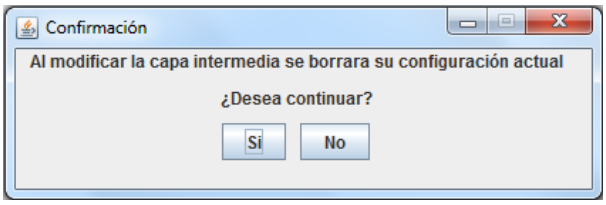

Ilustración 63: Modificar una arquitectura III.

Si el usuario decide no continuar con la modificación de la configuración de las capas ocultas, el simulador cerrará la ventana de confirmación y continuará en su estado actual. En caso contrario, si el usuario opta por confirmar la modificación de las capas ocultas, el simulador borrará la configuración actual confirmándole esto al usuario en el cuadro de información y habilitará de nuevo los botones para definir la configuración de las capas ocultas.

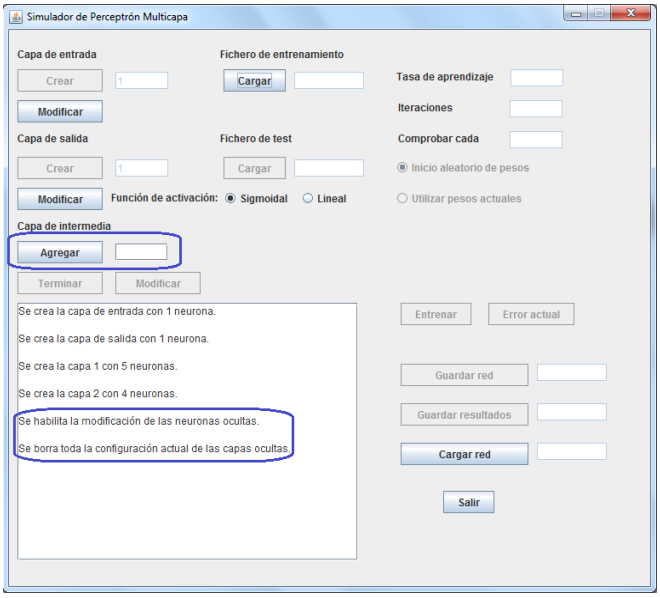

Ilustración 64: Modificar una arquitectura IV.

6.5. Función de activación de la salida

El simulador SPM el usuario presenta la opción de seleccionar la función de activación que se va a utilizar en las neuronas de salidas (ilustración 65). El usuario podrá seleccionar si prefiere aplicar una función de activación lineal o sigmoidal a las neuronas de salida, mientras que las neuronas ocultas siempre utilizaran una función de activación sigmoidal.

Podemos encontrar los botones para seleccionar la función de activación de la salida junto al botón de modificación de la capa de salida.

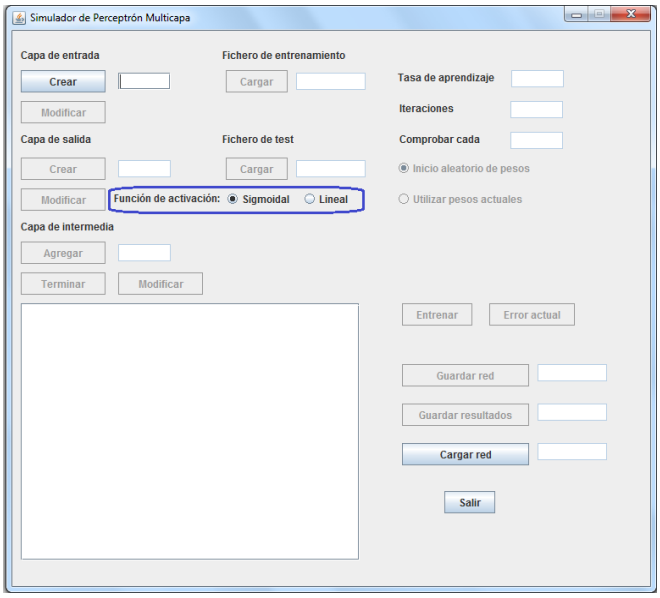

Ilustración 65: Función de activación de las neuronas de salida.

El SPM marcará por defecto la función de activación sigmoidal; dependerá del usuario decidir si la función de activación lineal es la adecuada o no.

La función de activación lineal será necesaria para ciertos casos, principalmente cuando el usuario no haya normalizado entre 0 y 1 las salidas deseadas de los ficheros de datos, dado que la función de activación sigmoidal hará que las salidas de la red solo puedan tomar valores entre 0 y 1.

6.6. Definir los parámetros de ejecución para el entrenamiento

En la esquina superior derecha del simulador se pueden observar una serie de parámetros para los que el usuario deberá asignar una serie de valores que van asociados al proceso del entrenamiento (ilustración 66).

Estos parámetros son: la tasa de aprendizaje, las iteraciones que durará el aprendizaje, cada cuántas iteraciones realizará el simulador una comprobación sobre la evolución del error y los valores que tomarán los pesos al comienzo del entrenamiento.

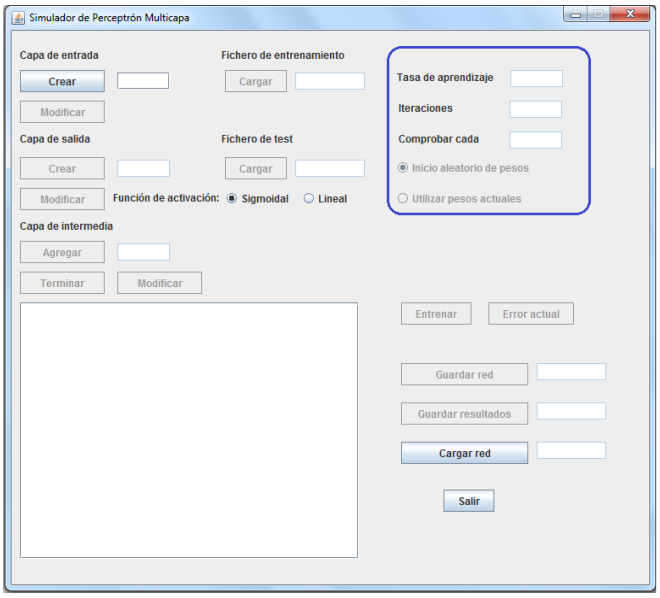

Ilustración 66: Parámetros de ejecución del entrenamiento.

6.6.1. Tasa de aprendizaje

El Perceptrón multicapa es un tipo de red de neuronas que se basa en realizar varias iteraciones sobre un conjunto de datos para los que conocemos sus salidas, comparando las salidas que obtiene con las conocidas.

El Perceptrón compara la salida obtenida con la salida deseada y corrige el valor actual de los pesos de la red para tratar de aproximar mejor la salida. El cambio en el peso es proporcional al gradiente del error, siendo α (tasa de aprendizaje) la constante de proporcionalidad, α.

La tasa de aprendizaje será un número real que toma valores entre 0 y 1, cuando la tasa sea 0 la red realizará las iteraciones del proceso de aprendizaje pero el error será constante e igual al error inicial, dado que la red va a tener siempre los mismos pesos.

6.6.2. Iteraciones

A continuación el usuario debe indicar cuantas iteraciones quiere realizar sobre el conjunto de datos durante el proceso de entrenamiento. Este valor tendrá que ser un número natural mayor que 1.

Si al finalizar el entrenamiento el usuario observa en la gráfica de evolución del error que la red aún puede adaptar sus pesos para obtener mejores resultados, es posible ampliar el número de iteraciones combinando este parámetro con la opción de “utilizar los pesos actuales”

Si se selecciona la opción “Utilizar los pesos actuales” y se vuelve a pulsar el botón de entrenar, el simulador continuará entrenando desde el punto donde finalizase el entrenamiento anterior y hará tantas iteraciones como el usuario indique en el parámetro de las iteraciones. De modo que si el usuario entrena una red 1000 iteraciones y tras hacerlo, marca la opción de utilizar los pesos actuales y vuelve a entrenar, equivaldrá a hacer un total de 2000 iteraciones.

6.6.3. Comprobaciones

Este parámetro se utiliza únicamente para generar la gráfica de evolución del error. El usuario tendrá que asignarle un valor entero entre 1 y el total de iteraciones. El simulador mostrará entonces en la gráfica de evolución del error el error medio sobre los ficheros de entrenamiento y test cada X iteraciones, siendo X el valor introducido por el usuario.

6.6.4. Valor inicial de los pesos

Con esta opción el usuario podrá decidir si al comienzo del entrenamiento se le asignan valores aleatorios a los pesos de la red o si se continúan utilizando los valores actuales.

Esta opción viene deshabilitado por defecto en el simulador SPM y su comportamiento por defecto es que los pesos se inicialicen de manera aleatoria. Una vez que los pesos ya tengan un valor, ya sea porque el usuario ha entrenado al menos una vez la red o haya cargado una red que haya sido guardada anteriormente, se habilitará esta opción para que el usuario pueda decidir.

Esta opción se deshabilita de nuevo en caso de que se modifique la arquitectura de la red, tomando de nuevo su valor por defecto.

6.7. Entrenar la red

Una vez que el usuario haya definido una arquitectura de red, cargado los ficheros de datos y definido los parámetros se habilitará en el simulador el botón de **Entrenar** para entrenar la red. También se habilitará en caso de cargar una red. Para entrenar la red el usuario debe pulsar el botón **Entrenar** (ilustración 67).

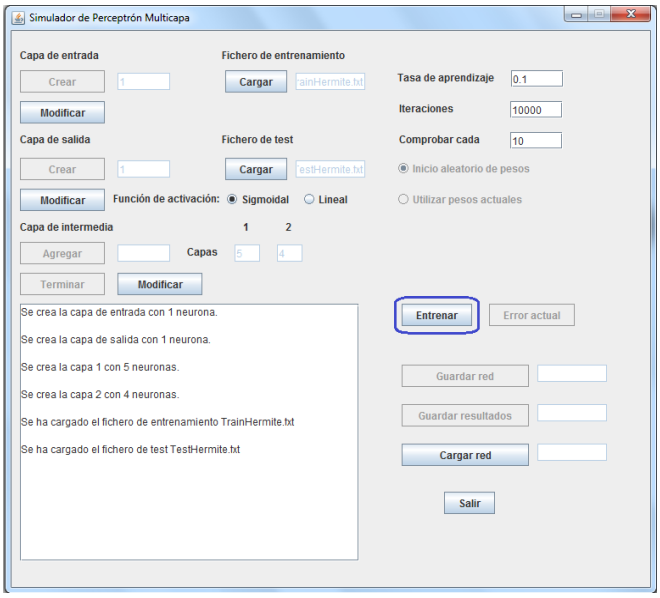

Ilustración 67: Entrenar una red.

Cuando el usuario decida llevar a cabo el entrenamiento y pulse el botón **Entrenar**, el simulador llevará a cabo una serie de comprobaciones sobre los datos que se han introducido en él. En caso de que alguna de las comprobaciones que lleva a cabo el simulador hallase algún dato incorrecto, el entrenamiento no se llevaría a cabo y el simulador devolvería al usuario un mensaje de error personalizado indicando el error. Algunos de los mensajes de error que puede recibir son los siguientes (ilustración 68 e ilustración 69):

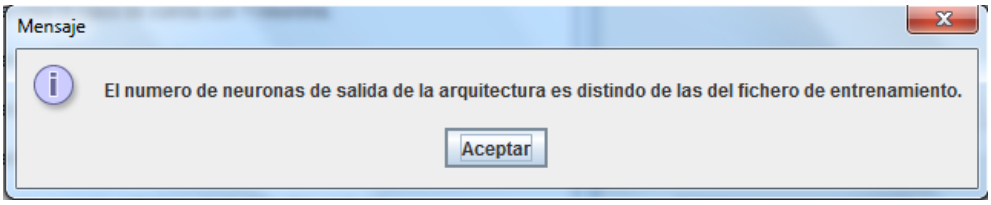

Ilustración 68: Entrenamiento, ejemplo de error I.

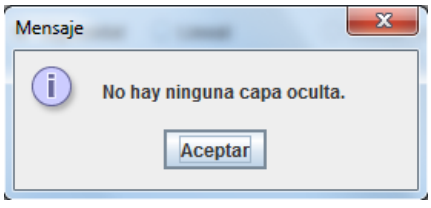

Ilustración 69: Entrenamiento, ejemplo de error II.

Si por el contrario todo esta correcto, el usuario verá como durante el proceso de entrenamiento se deshabilitan los demás botones del simulador y aparece una animación (ilustración 70).

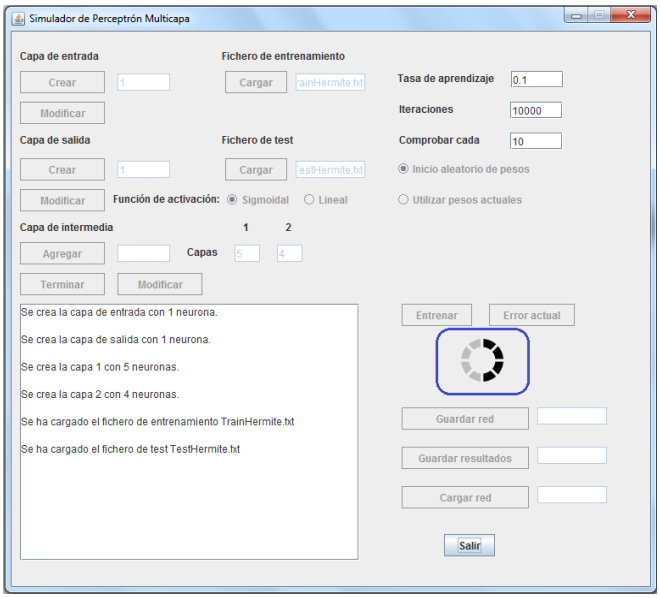

Ilustración 70: Proceso de entrenamiento.

Una vez que acabe el proceso de entrenamiento, se habilitan de nuevo los botones y desaparece la animación. El usuario podrá consultar el error medio en el cuadro de información de la esquina inferior izquierda (ilustración 71). También le aparecerá en una nueva ventana una gráfica con la evolución del error para los ficheros de entrenamiento y test durante el proceso de entrenamiento, la cual se explica a continuación en el capítulo 8 (Grafica de evolución del error) de este manual.

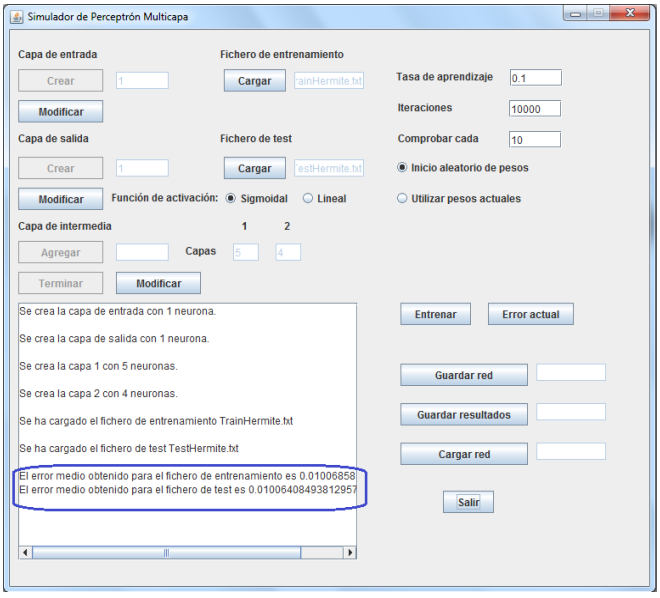

Ilustración 71: Error medio en el entrenamiento.

El error que se muestra tanto en el cuadro de información como en la gráfica de evolución del error y en el cálculo del error actual, es el error cuadrático medio por patrón, se calcula sobre todo el conjunto de datos y se viene dado por la siguiente expresión:

$$e = \frac{1}{N} * \sum_{i=1}^{N} (y_i - \tilde{y}_i)^2$$

Ecuación 19: Calculo del error cuadrático medio.

donde e es el error cuadrático medio, N el número de patrones, y_i la salida deseada para el patrón i y $\tilde{y}_i$ es la salida obtenida para dicho patrón.

6.8. Grafica de evolución del error

Cuando el proceso de entrenamiento se ha llevado a cabo correctamente, se muestra una gráfica en una nueva ventana en la que el usuario podrá ver la evolución del error a lo largo de las distintas iteraciones que se han llevado a cabo a lo largo del proceso de entrenamiento (ilustración 72).

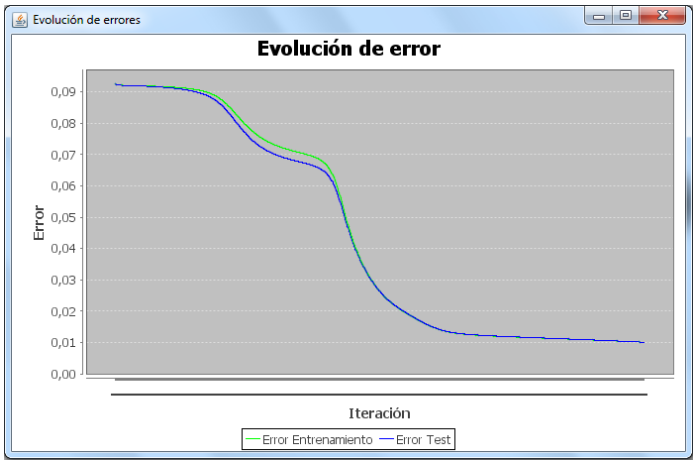

Ilustración 72: Grafica de evolución de error.

En esta gráfica, como se puede observar en la ilustración, el usuario podrá observar cómo han ido evolucionando los errores sobre los ficheros de entrenamiento y test a lo largo del proceso de aprendizaje.

El usuario podrá realizar diferentes acciones sobre la gráfica para ello deberá pulsar el botón secundario del ratón sobre la gráfica lo cual desplegara un menú con las opciones disponibles (ilustración 74). Estas son:

- Acercar la gráfica.

- Alejar la gráfica.
- Modificar la escala de la gráfica.
- Imprimir la gráfica.
- Guardar la gráfica.

Si el usuario desea hacer zoom sobre una zona en concreto de la gráfica sin hacer uso de las opciones del menú simplemente tendrá que seleccionar dicha área con el ratón (ilustración 73) o utilizar las opciones que se mostrarán en el menú (ilustración 74).

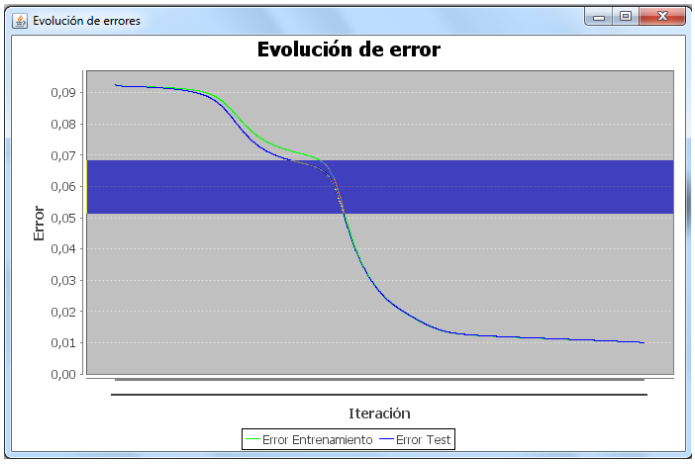

Ilustración 73: Grafica de evolución de error, zoom.

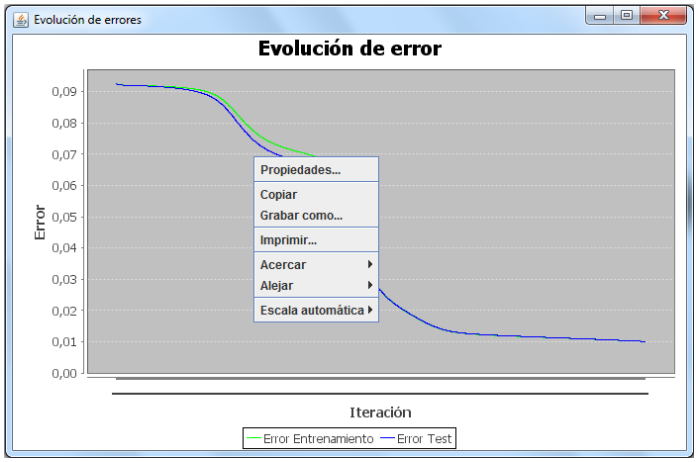

Ilustración 74: Grafica de evolución de error, menú.

6.9. Calculo del error actual

Una vez que el usuario tenga valores para los pesos, ya sea porque se ha llevado a cabo un entrenamiento o se haya cargado un fichero de red guardado anteriormente, se habilitará el botón de **Error actual** (ilustración 75).

Al pulsar este botón se muestra en el cuadro de información de la esquina inferior izquierda el error medio que se obtiene con la red actual sobre los conjuntos de datos.

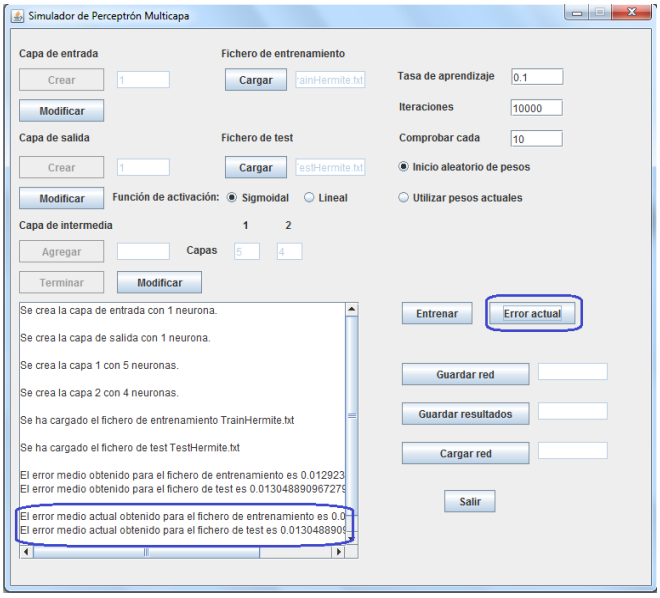

Ilustración 75: Error actual.

6.10. Guardar una red

El simulador ofrece al usuario la posibilidad de guardar la red con la que se está trabajando; para ello el usuario deberá pulsar el botón de **Guardar red** (ilustración 76). Al pulsar dicho botón aparece una ventana para seleccionar la dirección donde se va a guardar y el nombre del archivo donde se va a guardar.

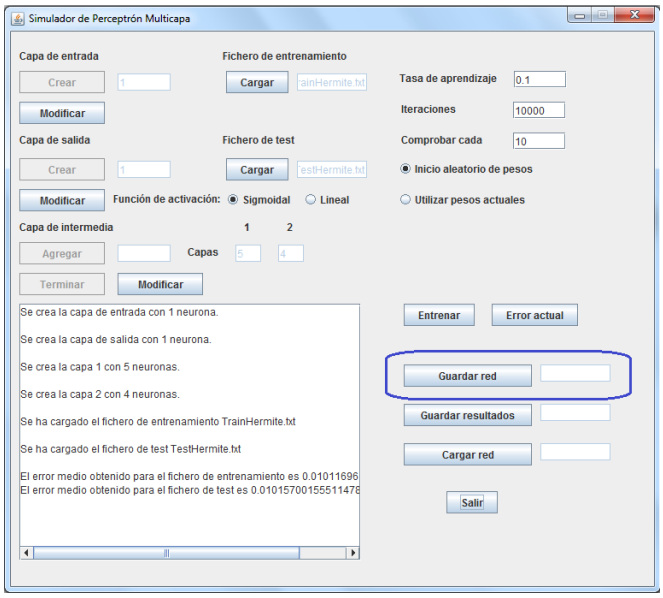

Ilustración 76: Guardar una red.

Al guardar una red no solo se guarda la arquitectura de ésta, sino que también se guardan los pesos y los umbrales asociados a la misma. Gracias a esto, el usuario podrá utilizar los valores obtenidos para implementar sus soluciones en cualquier otro sistema utilizando la red de neuronas que se ha obtenido.

El formato que tiene el fichero donde se guarda una red de neuronas es el siguiente:

- La primera línea indica con un número la arquitectura que tiene la red.
- La segunda línea indica que es el comienzo de los pesos de la red con la palabra "Pesos".
- La primera línea tras la etiqueta de "Pesos" está compuesta por los pesos que se aplican a las entradas respecto a las neuronas de entrada, siendo estos siempre 1.0 y habiendo tantos como neuronas de entrada.
- A partir de ahí, cada línea representa los pesos que salen de una neurona.
    - Las neuronas se van representando de arriba a abajo y de izquierda a derecha.
- Cuando se han representado todos los pesos, la siguiente línea es la etiqueta para el comienzo de los umbrales, "Umbrales".
- A partir de la etiqueta de "Umbrales", cada línea representará los umbrales de toda una capa, siendo la primera línea, los umbrales de la capa de entrada que siempre tomarán valor 0.0.
    - Las neuronas se van representando de arriba a abajo y de izquierda a derecha.

Para comprender mejor el formato se presenta el siguiente ejemplo que representa una red de neuronas con una neurona de entrada, dos capas ocultas con dos neuronas cada una y una neurona de salida (ilustración 77).

```
1 2 2 1                                 --> Neuronas que tiene cada capa
Pesos                                   --> Etiqueta de comienzo de pesos
1.0                                     --> Peso de la entrada a la neurona de entrada
4.156846908903794 12.231192640391054    --> Pesos de la neurona de entrada a la 1 y 2 de la primera capa oculta
-2.98815766636493 4.574184416786335     --> Pesos de la neurona 1 de la primera capa oculta a 1 y 2 de la segunda capa oculta
5.615398298824884 -4.202112182450956    --> Pesos de la neurona 2 de la primera capa oculta a 1 y 2 de la segunda capa oculta
6.938195444747009                       --> Peso de la neurona 1 de la segunda capa oculta a la neurona de salida
-6.608064186513758                      --> Peso de la neurona 2 de la segunda capa oculta a la neurona de salida
Umbrales                                --> Etiqueta de comienzo de umbrales
0.0                                     --> Umbrales de la neurona de entrada
-2.031479669672093 -1.6236308809427848  --> Umbrales de las neuronas 1 y 2 de la primera capa oculta
-2.5312118069810916 3.0691578390688643  --> Umbrales de las neuronas 1 y 2 de la segunda capa oculta
0.03865461614580363                     --> Umbral de la neurona de salida
```

Ilustración 77: Ejemplo comentado de fichero de guardado de red.

6.11. Guardar resultados de la red

El usuario tendrá la opción de guardar en un fichero las salidas que genera el simulador con una red previamente definida, cargada y/o entrenada para los datos de entrenamiento y test.

Para ello el usuario deberá pulsar el botón de **Guardar resultados** (ilustración 78).

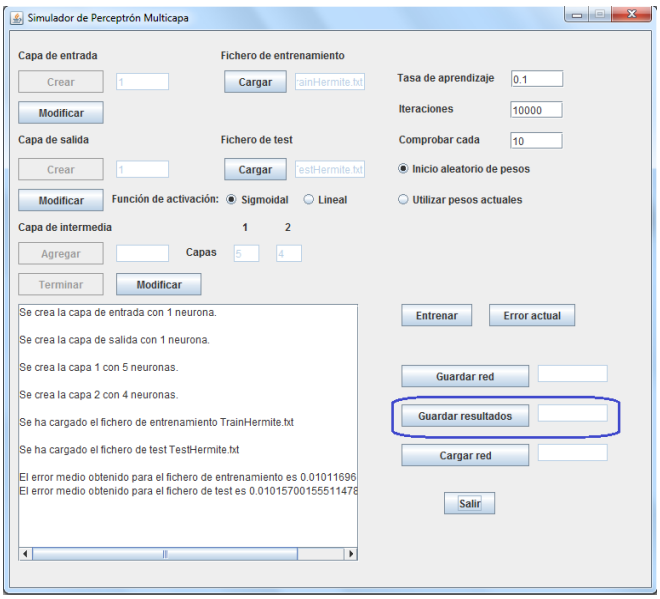

Ilustración 78: Guardar resultados.

Al hacerlo, al usuario se le solicitará que indique para qué fichero de datos desea guardar las salidas (ilustración 79).

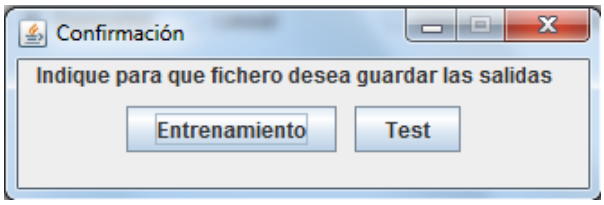

Ilustración 79: Guardar resultados, confirmación.

Una vez que seleccione el fichero para el cual desea guardar los resultados, al usuario le aparecerá una ventana en la que deberá seleccionar la ruta y nombre del fichero que se va a generar.

El fichero con los resultados que se va a generar tendrá el siguiente formato, en la primera línea se indica lo que representa cada columna de datos, siendo SD_i la nomenclatura para las salidas deseadas, y SR_i la nomenclatura para las salidas obtenidas.

A continuación se muestran dos ejemplos para una arquitectura con una única neurona de salida y para una arquitectura con múltiples neuronas de salida (ilustración 80 e ilustración 81).

```
SD0 SR0
0.001088437 0.035166452469686915
0.006078417 0.04052541676932961
0.015138645 0.048077684210262324
0.030777829 0.05906335307022017
0.056402394 0.07560424788428557
```

Ilustración 80: Formato del fichero de resultados (salida única).

```
SD0 SD1 SD2 SR0 SR1 SR2
1.0 0.0 0.0 0.9998943231899672 4.6943880411186704E-5 4.3766408435441295E-6
1.0 0.0 0.0 0.999896705442858 4.634113922394389E-5 4.387050401665945E-6
1.0 0.0 0.0 0.9749850901898272 0.016031356712942004 1.0287766939297394E-4
1.0 0.0 0.0 0.9999143811610932 4.103010438749333E-5 4.369366912149907E-6
0.0 0.0 1.0 5.937857192696713E-7 7.921308162039373E-4 0.9995531665293156
```

Ilustración 81: Formato del fichero de resultados (salidas múltiples).

6.12. Cerrar el simulador

Cuando el usuario desee cerrar el simulador bastará con pulsar el botón de **Salir** o pulsar el aspa de la esquina superior derecha, bien en la ventana de bienvenida (ilustración 82) o bien en la ventana principal del simulador (ilustración 83).

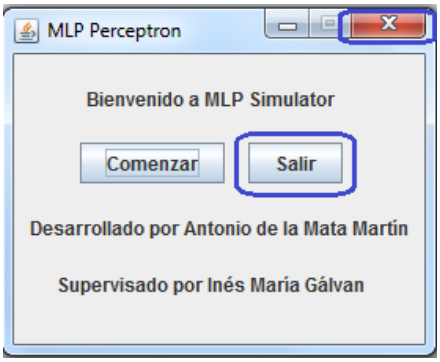

Ilustración 82: Salir del simulador (ventana de bienvenida).

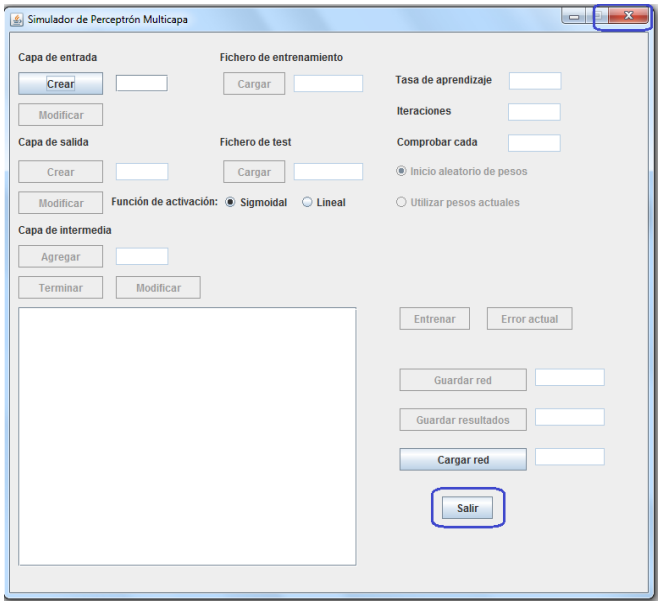

Ilustración 83: Salir del simulador (ventana principal).

Capítulo 7. Planificación y presupuesto

A largo de este capítulo se llevará a cabo un análisis del desarrollo del sistema. Este análisis estará compuesto por los siguientes puntos: un estudio de la planificación del proyecto, distinguiendo entre su planificación inicial y la que finalmente se ha llevado a cabo y la determinación de los costes que supondría el desarrollo del proyecto.

7.1. Planificación

Para un correcto desarrollo del sistema, al igual que sucede con la mayoría de los proyectos, es necesario realizar previamente un estudio en profundidad de las tareas que se han de llevar a cabo. Una vez conocidas las diferentes tareas y/o fases que se van a tener que realizar durante el desarrollo del proyecto, resulta de gran ayuda la elaboración de una planificación que estudie los plazos de realización de éstas, para así poder determinar el esfuerzo y tiempo que se ha de dedicar a cada una ellas.

A pesar de llevar a cabo una planificación inicial, es necesario revisarla frecuentemente para poder adaptar los plazos en función de los tiempos que finalmente se están llevando a cabo.

7.1.1. Definición de las tareas

Con carácter previo a la planificación del sistema, se llevó a cabo un estudio para la obtención de las diferentes tareas que se debían realizar a lo largo del desarrollo del mismo. El objetivo de este estudio era la identificación de estas tareas y la estimación del esfuerzo que supondría cada una de ellas para poder elaborar la planificación del desarrollo del sistema. Tras este estudio se identificaron las siguientes tareas:

1. Estudio del Perceptrón Multicapa.
2. Estudio de simuladores de RNAs y sistemas similares.
3. Análisis del sistema. Se llevará a cabo un análisis del sistema para reconocer las necesidades que ha de satisfacer y la complejidad que conllevan.
4. Elaboración del diseño del sistema.
5. Codificación y desarrollo del sistema.
6. Realización de pruebas y corrección de posibles fallos descubiertos en el sistema.
7. Elaboración de la documentación.

La fase de “Codificación del sistema” estará formada por diferentes sub-fases:

- Desarrollo de la interfaz. Se desarrolla la interfaz que se mostrará al usuario cuando ejecute el simulador, además de implementar las distintas interacciones que podrá realizar dicho usuario.
- Arquitectura. Se codifica el soporte para que el usuario pueda definir arquitecturas de red.
- Carga de ficheros de datos. Se desarrollan los métodos necesarios para poder cargar e interpretar ficheros de datos.
- Entrenamiento. Se implementan todos los métodos necesarios para llevar a cabo el proceso de aprendizaje de una red.
- Cálculo del error. Se codifican las operaciones necesarias para poder comprobar el error actual de la red sobre los patrones de datos.
- Gestión de ficheros internos del simulador. Se codifican los métodos para llevar a cabo el guardado y carga de redes y el guardado de resultados.

7.1.2. Planificación inicial

Una vez definidas las diferentes tareas y fases que se han de llevar a cabo durante el desarrollo del sistema, se procedió a la elaboración de una primera planificación, la cual se basó en la experiencia previa del autor del presente trabajo y en las fechas establecidas para el desarrollo del mismo.

Para facilitar la comprensión de la planificación del proyecto, se facilita el siguiente diagrama de Gantt (ilustración 84):

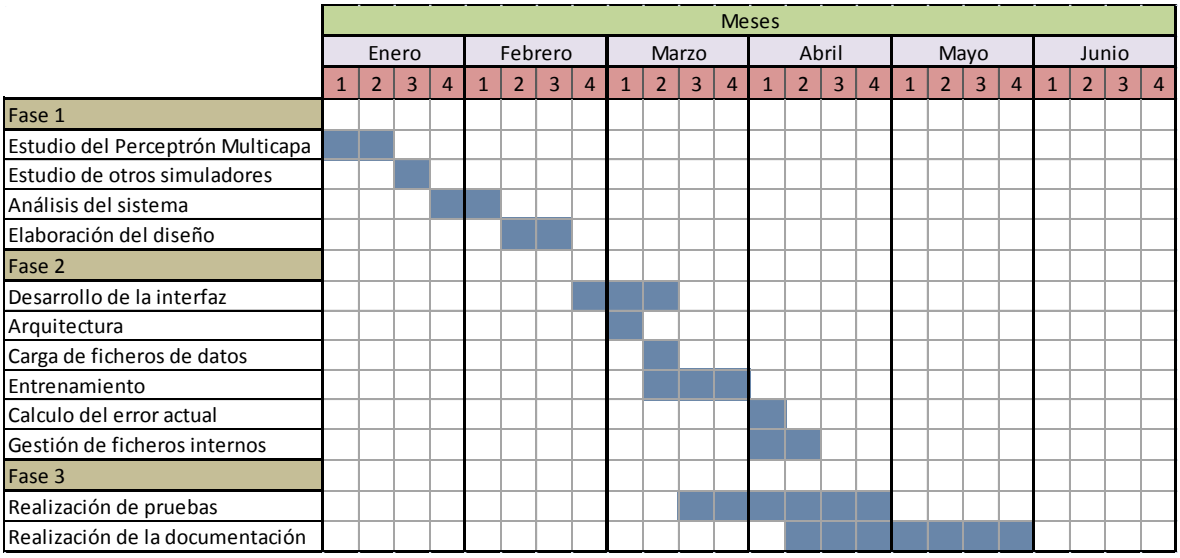

Ilustración 84: Diagrama de Gantt con la planificación inicial del proyecto.

En el diagrama anterior (ilustración 84) se puede observar el periodo de tiempo que se pretende dedicar en un principio al desarrollo de las distintas tareas que componen el presente proyecto. En el mismo se puede apreciar que hay tareas que se realizaron de manera

simultánea, esto se debe a que hay tareas que están estrechamente relacionadas y que el desarrollo de una tarea no siempre ocupa todas las horas de trabajo de una jornada. También se puede apreciar que se ha dejado un amplio margen de subsanación para poder hacer frente a los distintos imprevistos que puedan surgir durante el desarrollo del proyecto.

7.1.3. Planificación final

Como se mencionaba al comienzo de esta sección, el desarrollo de un proyecto es un proceso en constante evolución, lo cual implica que es necesario revisar frecuentemente su planificación para poder hacer frente a las diferentes circunstancias e imprevistos que puedan surgir a lo largo del desarrollo del mismo.

Una vez finalizado el desarrollo del proyecto se puede observar que su planificación ha sufrido algunos cambios. La planificación final del presente proyecto finalizó de la siguiente manera (ilustración 85):

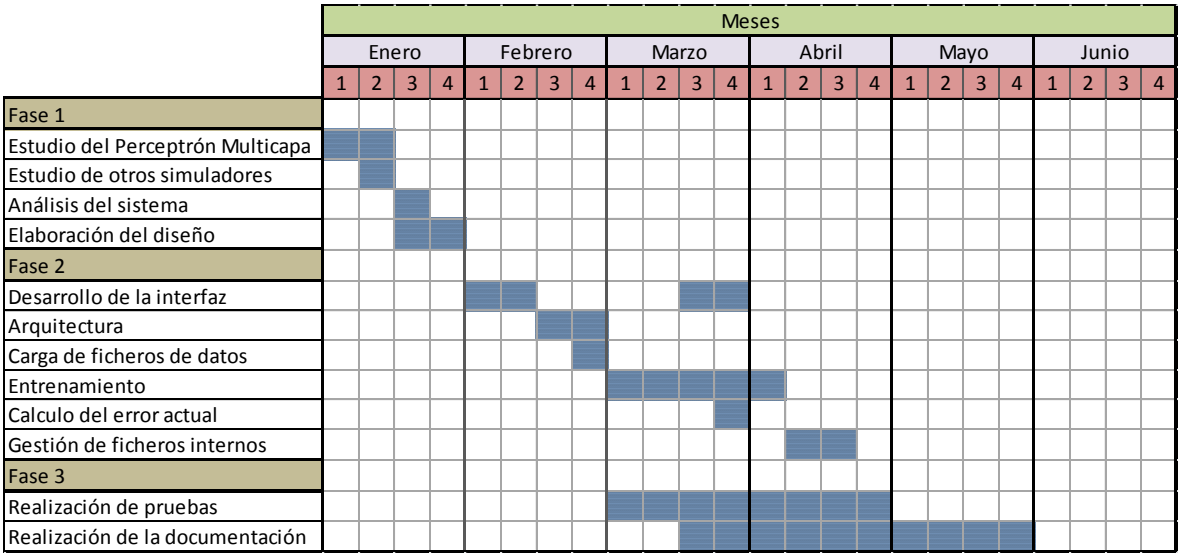

Ilustración 85: Diagrama de Gantt con la planificación final del proyecto.

En el diagrama anterior (ilustración 85) se pueden observar las distintas modificaciones que sufrió la planificación a lo largo del desarrollo del proyecto. Estas modificaciones, como se indicaba anteriormente, se deben a las distintas circunstancias a las que se hace frente durante el proceso de desarrollo del sistema. ´

Al igual que sucede en la mayoría de los proyectos, en el diagrama se puede observar que la tendencia general de las modificaciones en la planificación son para ampliar los plazos inicialmente previstos para cada una de las tareas. A pesar de las modificaciones, debido al margen de subsanación y adaptar frecuentemente la planificación en función de las necesidades del proceso de desarrollo, fue posible cumplir la fecha prevista para la finalización del proyecto.

7.2. Calculo de costes

En este apartado se procederá a determinar, detallar y explicar los costes asociados al desarrollo del proyecto en caso de que este fuera desarrollado de manera privada.

7.2.1. Determinación de los costes

Para poder determinar el coste total de un proyecto es necesario determinar los distintos costes que implica su desarrollo. Los costes a determinar en este proyecto son los salarios, los costes de amortización, gastos en materiales e impuestos.

A la hora de determinar los salarios se ha tenido en cuenta que al no dedicarse al desarrollo del sistema a jornada completa es necesario determinar el salario por hora, que se le habría pagado a un empleado. Para determinar dicho salario se ha tenido en cuenta que el salario medio en España [4]:

Concepto	Coste determinado
Salario medio Español	$22790 \frac{€}{año}$
Horas anuales	$52 \frac{semanas}{año} * 5 \frac{días}{semana} * 8 \frac{horas}{día} = 2080 \frac{horas}{año}$
Salario medio por hora	$\frac{22790 \frac{€}{año}}{2080 \frac{horas}{año}} = 10.95 \frac{€}{hora}$
Salario determinado por hora	$11 €$

Tabla 114: Determinación del salario por hora.

Una vez determinado el salario por hora que se va a aplicar al desarrollo del proyecto, se ha de determinar el coste total de los salarios durante el desarrollo del proyecto.

Persona encargada del desarrollo	Antonio de la Mata Martín
Horas dedicadas	$21 \text{ semanas} * 18 \frac{\text{horas}}{\text{semana}} = 378 \text{ horas}$
Coste por hora	$11 \frac{€}{hora}$
Coste total	$378 \, horas * 11 \frac{€}{hora} = 4158 €$

Tabla 115: Determinación del coste salarial del proyecto.

A continuación se determinarán esfuerzos acumulados, es decir, el número de horas invertidos en cada actividad.

Tarea	Esfuerzo dedicado
Estudio del Perceptrón Multicapa	15 horas durante 2 semanas
Estudio de otros simuladores	8 horas durante 1 semana
Análisis del sistema	20 horas durante 1 semana
Elaboración del diseño	25 horas durante 2 semanas
Desarrollo de la interfaz	35 horas durante 4 semanas
Arquitectura	25 horas durante 2 semanas
Carga de ficheros de datos	10 horas durante 1 semana
Entrenamiento	50 horas durante 5 semanas
Calculo del error actual	10 horas durante 1 semana
Gestión de ficheros internos	20 horas durante 2 semanas
Realización de pruebas	80 horas durante 8 semanas
Realización de la documentación	80 horas durante 10 semanas
Total	**378 horas durante 21 semanas**

Tabla 116: Esfuerzo dedicado por tareas.

Una vez que ya se ha estimado el coste salarial, se deben determinar los demás gastos asociados al desarrollo del proyecto. En primer lugar los gastos por amortización son los siguientes:

Coste	Gasto
Ordenador durante 21 semanas	50 €
Lugar de trabajo	400 €
Total	**450** €

Tabla 117: Gastos por amortización.

Por último los costes derivados del desarrollo del proyecto son:

Coste	Gasto
Material de oficina	40 €
Transporte	140 €
Total	**180** €

Tabla 118: Gastos derivados del proyecto.

7.2.2. Calculo de costes totales

Una vez determinados los diferentes costes asociados al desarrollo del proyecto y los esfuerzos que se han realizado para cada una de las tareas que lo componen, se puede calcular el coste total que implica desarrollar la herramienta (tabla 119). En la gráfica (ilustración 86) se representa el esfuerzo para cada tarea y el esfuerzo acumulado total a lo largo de las 21 semanas que duró el desarrollo del proyecto.

Coste	Gasto
Salarios	4158 €
Costes de amortización	450 €
Costes varios asociados al desarrollo	180 €
IVA	10%
Total sin IVA	4788 €
Total con IVA	**5266,8** €

Tabla 119: Costes totales del proyecto.

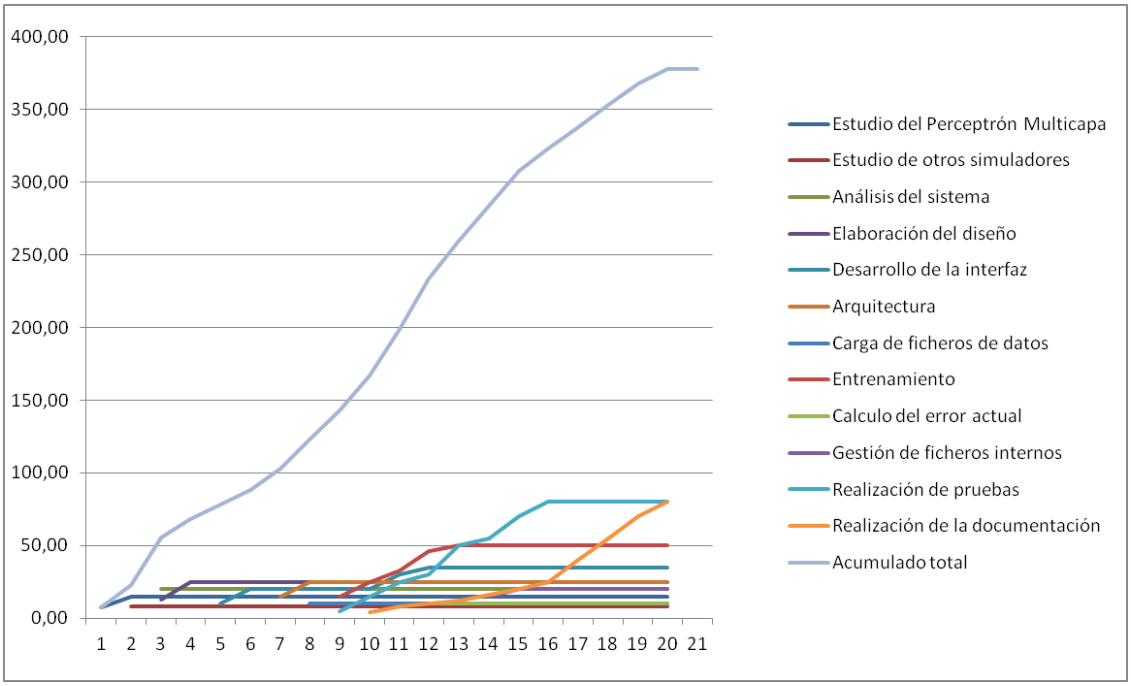

Ilustración 86: Grafico de esfuerzo acumulado.

Capítulo 8. Conclusions and future lines of work

Throughout this chapter, the different difficulties encountered during the development of the present paper will be presented, it also explains the conclusions which have been obtained, and finally, the possible future lines of work in case someone wishes to extend the tool.

8.1. Difficulties encountered

As it usually happens in most software development projects, throughout the development of the simulator, it was necessary to deal with the various difficulties encountered. The main difficulties encountered during the development of the simulator were:

- The complexity of handling three-dimensional data structures. In order to store the weights and other network information, it was necessary to find a data structure that was adequate for its storage and usage. The solution adopted was the use of two and three dimensional arrays whose structure is adapted to the data, but its use can be confusing since it is necessary to have to handle up to three indexes to retrieve the data.
- The difficulty of encoding the BackPropagation algorithm. The BackPropagation algorithm implementation turned out to be a complex process, since this algorithm makes use of many internal data structures of the simulator, such as the weights, the thresholds, the activations, the deltas and the architecture of the network. The use of these data structures combined with the complexity of the algorithm implementation made its encoding the most complicated point of the project.
- Incorrect results obtained in training. Once all the functionality for the training was finished, it was necessary to review all the parties involved in this process because the results were not the expected.
- The lack of experience working with interfaces. The little experience in designing and implementing interfaces caused that the simulator's interface had to be redesigned several times in order to suit the different changes that were made during the development of the simulator.
- The difficulty of finding a library for plotting graphs. In order to show the evolution of the error during the training it was necessary to search for a Java open source library. Like these charts are helpful to the user, it was considered an important part of implementation. Finally it was decided to use "JFreeChart" [5].

Despite these difficulties, the proper development of the simulator was possible.

8.2. Conclusions

Throughout this paper a RNA simulator for the Multilayer Perceptron has been documented and developed. This simulator allows the user to make use, in an easy and intuitive way, of the RNA architecture for solving problems most commonly used nowadays.

This simulator implements the different functionalities which involves the use of Multilayer Perceptron, allowing the user to define a network, load and manage data sets, defining the performance parameters of the training process and save (saving) the network and the results obtained.

When dealing with the various decisions and difficulties encountered during the development of the simulator, it has always been done taking into account that the main purpose of the project was the realization of a free and portable tool (preferably that did not require installation), because it is intended for academic and public use. Note the importance of the development of free tools and free software (GNU) in academic environments and countries with limited investments for education and research.

Finally, and despite the many difficulties that faced the project was completed satisfactorily, meeting the targets set at the beginning.

In this paper leveraging the development of the simulator, along with problems for validation of the system, it is also presented a real problem that uses the Multilayer Perceptron in order to predict the egg production in poultry farms. This problem has been treated as a time series prediction problem, whose objective is to perform a preliminary study to assess the ability of the network to predict the egg production on a farm in $t + 1$ and $t + 2$ from the birds' age and the production values of previous days, and to study how many previous production values are necessary.

The results of this preliminary study have shown that Multilayer Perceptron could be used to approach this problem, although it would be desirable to perform experiments with other batches of birds, to verify the network capacity to approach the problem.

8.3. Future lines of work

Due to the time constraints imposed by the current regulations for bachelor thesis, various features that would have been interesting to add to the simulator have not been carried out. Despite this, the tool is open to possible improvements:

- Cross-Validation. One of the possible future works in this simulator is to add the option of using the "cross-validation". Cross-validation is a statistical technique that is used for monitoring the error guaranteeing their independence from the partition of the dataset into training and test.

- Adding different network architectures. The Multilayer Perceptron is not always the best option for solving a problem, this simulator could be modified in order to integrate new network architectures and let the user select the architecture to be used.
- Display a graphical comparison between the desired and the obtained outputs. At the end of the training process, the simulator shows a graph with the evolution of the error (illustration 72). It would be interesting and useful for the user to add a second graph showing a comparison between the outputs obtained by the network and the desired ones, mainly in regression and prediction problems.
- Automating the process of normalization and data randomization. Thus, the user can standardize and randomize automatically from the simulator itself.
- Adding the possibility of selecting different languages. By modifying the interface and system messages it could be possible to add different languages in order to extend the use of the simulator to different countries.
- Extending the study of the egg production. It would be interesting to extend the problem of the egg production to $t + 3$ and $t + 4$, as well as extending the study to different geographical areas and other batches.

Definiciones y acrónimos

- RNA: Red de Neuronas Artificiales.
- ANN: Artificial Neural Networks.
- Topología: La topología de una red es el número de elementos de procesado (neuronas) que forman la red y las interconexiones existentes entre ellos.
- Ciclo: en el campo de las RNAs se dice que aparece un ciclo cuando la salida de una neurona es la entrada de otra, cuya salida es a su vez entrada de la primera, también pueden darse ciclos que incluyan un mayor número de neuronas.
- Acíclico: una red de neuronas se denomina acíclica si no presenta ciclos.
- Feed-forward: una red de neuronas se denomina feed-forward si las conexiones de sus neuronas no forman ciclos dirigidos.
- Activación: se denomina activación a las salidas de las neuronas.
- Mínimo global: valor de una función que es menor o igual a cualquier valor de la función dada. El mínimo absoluto es el menor de todos los valores.
- Mínimo local: valor de una función, que es menor que los valores de la función en puntos cercanos, pero que no es el menor de todos los valores.
- Ruido: se denomina ruido a toda señal o patrón no deseado o incorrecto que se mezcla con la señal útil.
- Identación: mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para distinguirlo mejor del texto adyacente, en los lenguajes de programación, la indentación es un tipo de notación secundaria utilizada para mejorar la legibilidad del código.

Bibliografía

[1] P. Isasi, I. M. Galván. *Redes de Neuronas Artificiales. Un Enfoque Práctico*. 2004. Pearson Educación S.A.

[2] *UCI Machine Learning Repository*. http://archive.ics.uci.edu/ml/.

[3] V.G. Narushin, C. Takma. *Sigmoid Model for the Evaluation of Growth and Production Curves in Laying Hens*. 2002. Biosystems Engineering. Páginas: 343-348.

[4] Instituto Nacional de Estadistica. http://www.ine.es/.

[5] Jfreechart. http://www.jfree.org/jfreechart/.

[6] H. Ahmadi, A. Golian. *Neural Network for Egg Production Curve*. 2008. Journal of Animal and Veterinary Advantages. Páginas: 1168-1170.

[7] B. Eckel. *Thinking in Java. 2007*. Prentice Hall.

[8] M. A. Weiss. *Data structures and algorithm analysis in Java*. 2007. Addison-Wesley.

[9] Gao Daqi, Yi Jan. *Classification methodologies of multilayer perceptrons with sigmoid activation functions*. 2005. Pattern Recognition volume 38.Páginas: 1469-1482.

[10] Lisa M. Belue, Kenneth W. Bauer Jr. *Determining input features for multilayer perceptrons*. 1995. Neurocomputing volume 7. Páginas: 111-121.

[11] Fionn Murtagh. *Multilayer perceptrons for classification and regression*.1991. Neurocomputing volumen 2. Páginas: 183-197.

[12] J. Hertz, A. Krogh, R. Palmer. *Introduction to the theory of Neural Computation*. 1991. Addison Wesley.

[13] P. Simpson. Artificial Neural Systems, Foundations, Paradigms, Applications and Implementations. 1990. Pergamon Press.

[14] S. K. Pal, S. Mitra. *Multilayer perceptron, fuzzy sets, and classification IEEE Transactions on Neural Networks*. 1992. Páginas: 683-697.

[15] M.W Gardner, S.R Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. Atmospheric Enviroment volume 32. Páginas: 2627-2636.

Apéndice I: Introducción.

Apéndice I.1. Introducción a las redes de neuronas artificiales

Las redes de neuronas artificiales surgen en el siglo XX a lo largo de los años cuarenta, cuando neurólogos como McCulloch y Pitts proponen los primeros modelos de redes de neuronas; Años más adelante Donald Hebb trabaja en diversas ideas sobre el aprendizaje neuronal que dan lugar a la regla de Hebb.

A finales de los años cincuenta aparecen los primeros modelos de redes neuronales que tienen aplicaciones prácticas en la industria, el Perceptrón simple y el ADALINE. En 1986 Rumelhart y otros autores presentaron la Regla Delta Generalizada para adaptar los pesos propagando los errores hacia atrás (retropropagación), para múltiples capas y funciones de activación no lineales. Es a partir de este momento cuando se produce un gran desarrollo en el campo de las redes de neuronas artificiales, surgiendo algunas de las redes más importantes, como son el Perceptrón Multicapa o las redes de neuronas de base radial.

Las Redes de Neuronas Artificiales (RNAs de aquí en adelante) son modelos matemáticos basados en sistemas biológicos naturales que se adaptan para poder ser simulados en computadores convencionales. Las RNAs son por lo tanto modelos de computación inspirados en las redes neuronales biológicas, que simulan la estructura de los sistemas nerviosos, los cuales se caracterizan por estar compuestos por neuronas conectadas entre sí. Al igual que sucede en las Redes de Neuronas Biológicas, la neurona artificial es la unidad elemental de procesamiento de las RNAs.

La estructura de la neurona artificial (ilustración 87) es similar a la estructura de la neurona biológica, teniendo varias entradas de información y computando una única salida que se corresponden respectivamente con las dendritas y el axón de la neurona biológica.

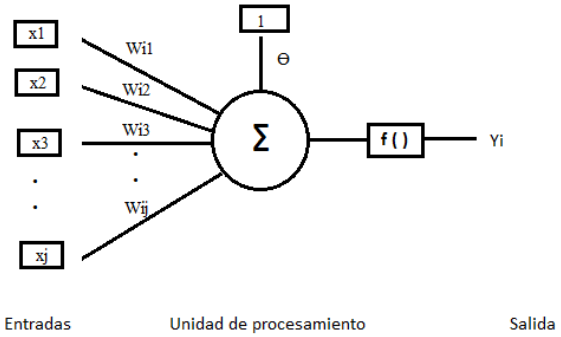

Ilustración 87: Estructura de una neurona artificial.

En la ilustración anterior se muestra la estructura de una neurona artificial, la cual cuenta con múltiples entradas que pueden proceder del mundo exterior o de otras neuronas y con una única salida cuyo valor se computa en función de las entradas de la misma.

Las RNAs se forman a partir de conjuntos de neuronas artificiales conectadas entre sí mediante arcos, que se denominan conexiones y que conectan las salidas de unas neuronas a las entradas de otras. Generalmente estas neuronas se organizan en capas a distintos niveles.

Existen numerosas arquitecturas de RNAs que responden a diversas clasificaciones, algunas de las RNAs más comunes clasificadas en función de su topología, es decir de los patrones de conexiones que presentan, son:

- Redes de propagación hacia adelante o acíclicas:
    - Monocapa: como el Perceptrón Simple o el ADALINE.
    - Multicapa: como el Perceptrón Multicapa o las Redes de Base Radial.
- Redes recurrentes: son redes que presentan al menos un ciclo como las redes de Hopfield o la máquina de Boltzmann.

Las RNAs tienen diferentes maneras de llevar a cabo el proceso de aprendizaje, pueden hacerlo de manera supervisada o no supervisada. El aprendizaje supervisado es aquel en el cual se requiere que el conjunto de datos de entrada este previamente clasificado o cuya respuesta objetivo sea conocida, mientras que en el aprendizaje no supervisado el conocimiento de esta salida no es necesario.

Las RNAs se utilizan para la resolución de problemas de clasificación, regresión o predicción de series temporales, entre otros. El motivo por el que se utilizan es que las RNAs se caracterizan por disponer de un aprendizaje adaptativo, debido a esto son capaces de aprender a realizar tareas en base a un entrenamiento específico y adaptarse más adelante a cambios recibidos como nuevas informaciones. Se caracterizan, además, porque son capaces de aprender relaciones a partir de ejemplos o patrones que representan el problema.

Apéndice I.2. Objetivos del proyecto

Con el presente Proyecto de Fin de Grado se pretende, como principal objetivo, construir un simulador de redes de neuronas para la red del Perceptrón Multicapa, para así facilitar una herramienta para uso docente en el campo de las redes neuronales. Este simulador incorporará todos los aspectos involucrados en la creación y entrenamiento del Perceptrón Multicapa. Las funcionalidades que deberá realizar el simulador son:

- Definir y modificar una arquitectura.
- Definir la función de activación de las neuronas de salida de la red.
- Cargar ficheros de entrenamiento y test.
- Definir los parámetros para el entrenamiento de una red.

- Entrenar una red.
- Almacenar una arquitectura de red.
- Cargar una arquitectura de red.
- Almacenar los resultados de la red.

Además de este objetivo principal, durante la planificación y desarrollo del trabajo también se han tenido en cuenta otros objetivos. Se pretende que el simulador sea una herramienta sencilla, fácil de utilizar y que no requiera de un amplio conocimiento previo sobre redes de neuronas, que tenga una interfaz cómoda y agradable y que no requiera de una instalación dificultosa, facilitando así al usuario su uso y portabilidad.

Apéndice I.3. Motivación del proyecto

Las RNAs son una rama de la Inteligencia Artificial que la Informática utiliza habitualmente como una de las posibles herramientas para proporcionar soluciones eficientes a problemas de diversos tipos como pueden ser la clasificación, aproximación o predicción en numerosos y diversos campos de estudio, entre ellos en medicina, economía, ingeniería, etc.

A lo largo de los años tras realizar diversos trabajos basados en estos problemas siempre encontraba la misma complicación al intentar aplicar las RNAs. Dicha complicación es la dificultad para encontrar herramientas gratuitas y eficientes que resulten intuitivas, sencillas de utilizar y que no requirieran de una compleja instalación.

Ante esta situación surge la idea y la motivación para desarrollar una nueva herramienta que cumpliese con dichos requisitos, para así proporcionar un medio cómodo, eficiente y manejable para desarrollar soluciones basadas en las RNAs y en particular utilizando el Perceptrón Multicapa. Esta herramienta proporcionaría acceso a dichas soluciones, pero no solo lo haría para personas versadas en el campo de las RNAs, sino que en la medida de lo posible facilitaría que personas sin amplios conocimientos previos acerca de su estructura y funcionamiento pudieran tener acceso a éstas, aprovechándose así de las ventajas que las RNAs pueden ofrecer a la resolución de estos problemas.

Para la puesta en marcha de un nuevo simulador, como para el desarrollo de cualquier aplicación, es necesario llevar a cabo un estudio previo de herramientas semejantes, lo cual facilita un mayor conocimiento acerca de las funcionalidades que ésta ha de llevar a cabo. Actualmente, existen diversas herramientas que permiten la simulación de arquitecturas de redes neuronales, y concretamente del Perceptrón Multicapa.

SNNS y su versión en Java JavaNNS son simuladores de redes de neuronas de libre acceso que permiten la generación de arquitecturas y aprendizaje para obtener una serie de resultados en función del tipo de arquitectura y los patrones establecidos. El SNNS en su versión para Linux funciona de manera correcta e intuitiva para el Perceptrón Multicapa, sin embargo resulta complejo de utilizar para otras arquitecturas de red y además tiene la desventaja de que es una herramienta con un proceso de instalación muy complejo. Por el contrario, la principal

ventaja de su versión en Java es que requiere de un proceso de instalación sencillo y que no requiere un entorno con grandes especificaciones para su ejecución, no obstante resulta poco intuitivo en su uso y su gran desventaja respecto al Perceptrón Multicapa es que inicializa los pesos de la red a cero, lo cual hace que en muchas ocasiones el aprendizaje no se consiga con éxito.

WEKA es una aplicación de libre acceso que proporciona a los usuarios las herramientas necesarias para resolver problemas de clasificación y aproximación o regresión mediante el uso de diferentes metodologías, como pueden ser árboles de clasificación, las redes bayesianas o el Perceptrón Multicapa entre otras. WEKA es una herramienta ampliamente utilizada que cuenta con un amplio reconocimiento en el campo de estudio de la Inteligencia Artificial, pero en lo referente a su simulador del Perceptrón Multicapa cuenta con ciertas deficiencias. Al igual que sucediese con el SNNS y JavaNNS su uso no es intuitivo, además su mayor desventaja es que no muestra la evolución del error a lo largo del proceso de aprendizaje, solo muestra el error final lo cual dificulta la definición de los parámetros para el aprendizaje.

NeuroSolutions es un software destinado principalmente a la resolución problemas con una complejidad elevada mediante diferentes tipos de metodologías, entre ellas las RNAs. Esta herramienta presenta una interfaz cómoda e intuitiva, pero su elevado número de opciones dificulta su uso a los usuarios más inexpertos, ya que al ofrecer una amplia variedad de metodologías, tanto del campo de las redes neuronales como de otros campos de estudio, el empleo de una metodología específica requiere un amplio conocimiento acerca de las diferentes alternativas que ofrece esta herramienta. No obstante, el principal inconveniente es su elevado precio, lo que no permite su accesibilidad a todo el público.

Neural Network Toolbox es una librería de Matlab que permite la simulación de diferentes arquitecturas de redes neuronales, pero al igual que sucede con NeuroSolutions tiene un coste elevado debido a que requiere la compra de licencias para el uso de Matlab, además requiere de conocimientos de programación en Matlab por parte del usuario.

También existen librerías ampliamente utilizadas y validadas como FastArtificial Neural Nerwork (FANN) en código C o Neuroph en Java, pero el uso de estas librerías implica que el usuario debe realizar un trabajo de programación previa.

Apéndice I.4. Estructura de la memoria

El presente documento conforma la base teórica y la estructura sobre la que se ha realizado el simulador. El documento muestra la siguiente estructura:

El capítulo actual hace una breve introducción a las RNAs, a la par que contesta a dos preguntas importantes, ¿Qué se pretende conseguir mediante la realización de este trabajo? y ¿Por qué se toma dicha iniciativa?

En el segundo capítulo, se detallan los fundamentos teóricos del Perceptrón Multicapa, sobre los cuales se cimienta el simulador.

A lo largo del tercer capítulo se realiza un análisis en profundidad del sistema, en el que se especifican los requisitos y restricciones que éste deberá satisfacer y un estudio de las posibles alternativas a la solución por la que se ha optado.

En el cuarto capítulo, se estudia el diseño del sistema, incluyendo la arquitectura, los subsistemas y las distintas clases que conforman la aplicación.

En el quinto capítulo, se especifican una serie de pruebas experimentales sobre diferentes dominios en las que se podrá analizar el correcto funcionamiento del simulador, así como los resultados obtenidos por éste.

En el sexto capítulo, se incluye el manual de usuario del simulador, el cual se proporcionará junto a éste para facilitar así su uso. El manual detalla cómo hacer uso de las distintas funcionalidades y opciones que ofrece el simulador.

En el capítulo séptimo, se detalla la planificación del desarrollo de la aplicación, la cual incorpora un análisis de las diferentes fases del proyecto y el presupuesto para su desarrollo.

En el capítulo octavo, se explican las diferentes conclusiones que se han obtenido durante el desarrollo de este trabajo, así como las futuras posibles líneas de trabajo en caso de que se desease continuar ampliando las funcionalidades de este simulador. Se incluye también un análisis de las dificultades que se han encontrado durante su desarrollo.

Después del octavo y último capítulo se han incluido una serie de definiciones para facilitar la compresión del documento y la bibliografía en la que se ha basado.

Por último al final del documento se han incluido dos apéndices que muestran los capítulos de la introducción y conclusiones en español.

Apéndice II: Conclusiones y futuras líneas de trabajo.

A lo largo del presente capítulo se explicarán las dificultades que se han encontrado durante la realización presente Trabajo de Fin de Grado, también se expondrán las conclusiones que se han obtenido, y finalmente, posibles líneas futuras de trabajo en caso de que se desease ampliar la herramienta.

Apéndice II. 1. Dificultades encontradas

Al igual que sucede en la mayoría de los proyectos de desarrollo de Software, a lo largo del desarrollo del simulador fue necesario hacer frente a las distintas dificultades que se fueron hallando. Las principales dificultades que se encontraron durante el desarrollo del simulador son:

- Complejidad en el manejo de estructuras de datos tridimensionales. Para almacenar los pesos y distinta información de la red fue necesario buscar una estructura de datos que fuese adecuada su almacenamiento y utilización. La solución que se tomó fue el uso de Arrays bidimensionales y tridimensionales cuya estructura se adaptaba a los datos, pero cuyo uso puede resultar confuso al tener que manejar hasta tres índices para recuperar los datos.
- Dificultad de la codificación del algoritmo de RetroPropagación. La implementación del algoritmo de RetroPropagación resultó ser un proceso complejo, debido a que este algoritmo hace uso de numerosas estructuras de datos internas del simulador, como son los pesos, los umbrales, las activaciones, los deltas y la propia arquitectura de la red. El uso de todas estas estructuras de datos unidas a la complejidad del algoritmo hizo que su implementación fuera el punto más complicado del proyecto.
- Resultados incorrectos en el entrenamiento. Una vez finalizado toda la funcionalidad correspondiente al entrenamiento, fue necesario revisar todas las partes que participaban en dicho proceso debido a que los resultados que aportaban, no eran los esperados.
- Falta de experiencia trabajando con interfaces. La falta de experiencia a la hora de diseñar e implementar interfaces hizo que la interfaz del simulador tuviera que ser rediseñada numerosas veces para adaptarse a los distintos cambios que se realizaron a lo largo del desarrollo del simulador.
- Dificultad para encontrar una librería para gráficas. Para mostrar la evolución del error una vez se había realizado el entrenamiento era necesario la búsqueda de una librería de acceso libre en Java para realizar gráficas. Estas gráficas son de gran ayuda al usuario, por lo que se consideró importante e imprescindible su implementación. Finalmente se optó por la utilización de “JFreeChart” [5] una librería de acceso libre.

A pesar de estas dificultades fue posible el correcto desarrollo del simulador.

Apéndice II. 2. Conclusiones

A lo largo del presente Trabajo de Fin de Grado se ha documentado y desarrollado un simulador de RNAs para la arquitectura del Perceptrón Multicapa. Este simulador permite al usuario hacer uso de manera sencilla e intuitiva de una de las arquitecturas de RNAs para la resolución de problemas que más se utiliza en la actualidad.

El simulador implementa las diferentes funcionalidades que implica el uso del Perceptrón Multicapa, permitiendo al usuario definir una red, cargar y manejar conjuntos de datos, definir los parámetros de ejecución del proceso de entrenamiento y guardar la red y los resultados obtenidos.

A la hora de hacer frente a las distintas decisiones y dificultades encontradas a lo largo del desarrollo del simulador, siempre se ha tenido en cuenta que el principal propósito del proyecto era la realización de una herramienta gratuita y portable (a ser posible que no requiriese instalación), debido a que está orientado para su uso académico y público. Cabe destacar la importancia que tiene el desarrollo de herramientas gratuitas o Software libre (GNU) para entornos académicos y países que cuentan con inversiones reducidas para educación e investigación.

Finalmente y pese a las diferentes dificultades a las que se tuvo que hacer frente el proyecto se completó de manera satisfactoria, cumpliendo con los objetivos marcados en un principio.

En este trabajo de fin de grado aprovechando el desarrollo del simulador, además de los problemas y datos utilizados para la validación del software, se ha abordado también un problema real que consiste en utilizar el Perceptrón Multicapa para predecir la producción de huevos en granjas avícolas. Dicho problema se ha tratado como un problema de predicción de series temporales, cuyo objetivo ha sido realizar un estudio preliminar para observar la capacidad de la red para predecir la producción de huevos en una granja en $t+1$ y en $t+2$ a partir de la edad del ave y de valores de producción de días anteriores, así como estudiar cuántos valores anteriores de producción eran necesarios.

Los resultados obtenidos en este estudio preliminar han mostrado que el Perceptrón Multicapa podría utilizarse para abordar este problema, aunque sería conveniente realizar experimentación con otros lotes de aves, para verificar la capacidad de la red para abordar el problema.

Apéndice II. 3. Futuras líneas de trabajo

Debido a las limitaciones de tiempo impuestas por la presente normativa para los Trabajos de Fin de Grado no se han llevado a cabo distintas funcionalidades que hubiera sido interesante añadir al simulador. A pesar de esto, esta herramienta está abierta a posibles mejoras:

- Validación cruzada. Una de las posibles futuras líneas de trabajo en este simulador consiste en añadir la opción de utilizar la "validación cruzada". La validación cruzada es una técnica estadística que se utiliza para comprobar la evolución del error garantizando su independencia respecto de la partición del conjunto de datos en entrenamiento y test.
- Añadir diferentes arquitecturas de Red. Debido a que el Perceptrón Multicapa no es siempre la mejor opción para la resolución de un problema, se podría modificar el presente simulador para integrar nuevas arquitecturas de red y que al comienzo del mismo dejase al usuario seleccionar la arquitectura de red que desea utilizar.
- Muestra de gráfica comparativa entre la salidas deseadas y las salidas obtenidas. Al finalizar el proceso de entrenamiento, el simulador muestra una gráfica con la evolución del error (ilustración 72). Sería interesante y de gran utilidad para el usuario añadir una segunda gráfica que muestre una comparativa entre las salidas obtenidas por la red y las salidas deseadas, fundamentalmente en problemas de regresión y predicción.
- Automatización del proceso de normalización y aleatorización de los datos. De este modo, el usuario pueda normalizar y aleatorizar de manera automática desde el propio simulador.
- Añadir la opción de seleccionar el idioma. Mediante la modificación de la interfaz y los mensajes del sistema para que se pudiese seleccionar el idioma del simulador se podría ampliar su uso a diferentes países.
- Ampliar el estudio de la producción de huevos. Haciendo un estudio de la producción de huevos en $t+3$ y $t+4$, así como extendiendo el estudio a diferentes areas geográficas y otros lotes de aves.